IMA Journal of Numerical Analysis (2024) **00**, 1–31
https://doi.org/10.1093/imanum/drad099

Solving nonlinear ODEs with the ultraspherical spectral method

Ouyuan Qin and Kuan Xu*
School of Mathematical Sciences, University of Science and Technology of China, 96 Jinzhai Road, Hefei, Anhui 230026, China
*Corresponding author: kuanxu@ustc.edu.cn

[Received on 23 February 2023; revised on 27 October 2023]

We extend the ultraspherical spectral method to solving nonlinear ordinary differential equation (ODE) boundary value problems. Naive ultraspherical Newton implementations usually form dense linear systems explicitly and solve these systems exactly by direct methods, thus suffering from the bottlenecks in both computational complexity and storage demands. Instead, we propose to use the inexact Newton–GMRES framework for which a cheap but effective preconditioner can be constructed and a fast Jacobian-vector multiplication can be effected, thanks to the structured operators of the ultraspherical spectral method. The proposed inexact Newton–GMRES–ultraspherical framework outperforms the naive implementations in both speed and storage, particularly for large-scale problems or problems whose linearization has solution-dependent variable coefficients in higher-order terms. Additional acceleration can be gained when the method is implemented with mixed precision arithmetic.

Keywords: spectral method; nonlinear ODEs; boundary value problems; Chebyshev polynomials; ultraspherical polynomials.

1. Introduction

In this article, we extend the ultraspherical spectral method (Olver & Townsend, 2013) to solving the nonlinear ordinary differential equation (ODE) boundary value problem

$$\mathcal{F}(u) = 0, \text{ s.t. } \mathcal{N}(u) = 0,$$

where $\mathcal{F}$ is a nonlinear differential operator on $u(x)$. The solution $u(x)$ is a univariate function of the independent variable $x \in [-1, 1]$. The functional constraint $\mathcal{N}$ contains linear or nonlinear boundary conditions or side constraints of other types, such as interior point conditions, global constraints, etc.

In the final paragraph of Olver & Townsend (2013), the authors briefly discuss the possibility of solving nonlinear differential equations by the ultraspherical spectral method and caution the loss of bandedness in the multiplication operators as a threat to the sparsity of the linear system and, therefore, to the exceptional speed of the ultraspherical spectral method. It has been a decade since Olver & Townsend (2013) was published. While there have been some preliminary implementations of the ultraspherical spectral method for nonlinear problems (Driscoll *et al.*, 2014; Olver, 2019; Burns *et al.*, 2020), it seems that no substantial progress has been made towards this extension, particularly for large-scale problems. This paper aims to fill this long-overdue gap.

What we find in this study is that the loss of bandedness in multiplication operators can be remediated surprisingly easily with the inexact Newton–GMRES framework, a proper preconditioner and a correct way to carry out the Jacobian-vector multiplication. The proposed framework is easy to implement and can largely restore the speed of the original ultraspherical spectral method without compromising on the accuracy and stability.

We begin by first setting up the inexact Newton–GMRES–ultraspherical (INGU) framework in Section 2. We discuss the fast application of the truncated Fréchet operators in Section 3 and the preconditioning in Section 4. Section 5 describes a few possibilities to further speed up the computation, including our mixed precision implementation. Numerical experiments are shown in Section 6 before we close in the final section.

Throughout this article, calligraphy fonts are used for operators or infinite matrices and bold fonts for infinite vectors, whereas functions, truncations of operators, infinite matrices and infinite vectors are denoted by normal fonts. We denote the infinite identity operator and the $n \times n$ identity matrix by $\mathcal{I}$ and I_n, respectively.

2. The INGU framework

To make our discussion uncluttered, we give a quick review of the very essence of the ultraspherical spectral method in Section 2.1, which we could have dispensed with, but at a cost of annoyingly frequent reference to Olver & Townsend (2013). We discuss the linearization, truncation and adaptivity in Section 2.2, which yield a primitive ultraspherical-based Newton method. The inexact Newton condition enabled by GMRES and the three popular global Newton variants briefly reviewed in Section 2.3 and Section 2.4, respectively, lead to the prototype INGU method given in Section 2.5.

2.1 *Ultraspherical spectral method*

The ultraspherical spectral method solves the linear ODE

$$\begin{aligned} \mathcal{L}u &= f \\ \text{s.t. } \mathcal{B}u &= c \end{aligned} \tag{2.1}$$

by approximating the solution with a Chebyshev series $u(x) = \sum_{j=0}^{\infty} u_j T_j(x)$, where $T_j(x)$ is the Chebyshev polynomial of degree j. Here, $\mathcal{L}$ is an Nth order linear differentiation operator

$$\mathcal{L} = a^N(x)\frac{\mathrm{d}^N}{\mathrm{d}x^N} + \ldots + a^1(x)\frac{\mathrm{d}}{\mathrm{d}x} + a^0(x), \tag{2.2}$$

and $\mathcal{B}$ contains N linear functionals of boundary conditions. Once the coefficients u_j are known, the solution $u(x)$ is identified by the coefficient vector $\boldsymbol{u} = (u_0, u_1, u_2, \ldots)^\top$.

For $\lambda \geq 1$, $\mathrm{d}^\lambda/\mathrm{d}x^\lambda$ is replaced by the λth-order differentiation operator

$$\mathcal{D}_\lambda = 2^{\lambda-1}(\lambda-1)!\begin{pmatrix} \overbrace{0 \;\cdots\; 0}^{\lambda \text{ times}} & \lambda & & & \\ & & \lambda+1 & & \\ & & & \lambda+2 & \\ & & & & \ddots \end{pmatrix},$$

mapping the Chebyshev T coefficients to the ultraspherical $C^{(\lambda)}$ coefficients.[1]

[1] In Olver & Townsend (2013), $\mathcal{D}_0 = \mathcal{D}_1$, while in this paper $\mathcal{D}_1$ maps from Chebyshev T to $C^{(1)}$ and $\mathcal{D}_0 = \mathcal{I}$, i.e., the identity operator, for notational consistency.

When $a^0(x) = \sum_{j=0}^{\infty} a_j T_j(x)$ is variable, the action of $a^0(x)$ on $u(x)$ is represented by a Toeplitz-plus-Hankel-plus-rank-1 multiplication operator

$$\mathcal{M}_0[a^0] = \frac{1}{2}\left[\begin{pmatrix} 2a_0 & a_1 & a_2 & a_3 & \cdots \\ a_1 & 2a_0 & a_1 & a_2 & \ddots \\ a_2 & a_1 & 2a_0 & a_1 & \ddots \\ a_3 & a_2 & a_1 & 2a_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots \\ a_1 & a_2 & a_3 & a_4 & \cdots \\ a_2 & a_3 & a_4 & a_5 & \iddots \\ a_3 & a_4 & a_5 & a_6 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix}\right].$$

If any of $a^\lambda(x) = \sum_{j=0}^{\infty} a_j^\lambda C_j^{(\lambda)}(x)$ for $\lambda > 0$ is not constant, the differential operator $\mathcal{D}_\lambda$ should be pre-multiplied by the multiplication operator $\mathcal{M}_\lambda[a^\lambda]$, which represents the product of two $C^{(\lambda)}$ series. The most straightforward way to generate $\mathcal{M}_\lambda[a^\lambda]$ (Townsend, 2014, §6.3.1) is to express it by the series

$$\mathcal{M}_\lambda[a^\lambda] = \sum_{j=0}^{\infty} a_j^\lambda \mathcal{M}_\lambda\big[C_j^{(\lambda)}\big], \tag{2.3a}$$

where $\mathcal{M}_\lambda\big[C_j^{(\lambda)}\big]$ is obtained by the three-term recurrence relation

$$\mathcal{M}_\lambda\big[C_{j+1}^{(\lambda)}\big] = \frac{2(j+\lambda)}{j+1}\mathcal{M}_\lambda[x]\mathcal{M}_\lambda\big[C_j^{(\lambda)}\big] - \frac{j+2\lambda-1}{j+1}\mathcal{M}_\lambda\big[C_{j-1}^{(\lambda)}\big], \quad j \geq 1 \tag{2.3b}$$

with $\mathcal{M}_\lambda\big[C_0^{(\lambda)}\big] = \mathcal{I}$, $\mathcal{M}_\lambda\big[C_1^{(\lambda)}\big] = 2\lambda\mathcal{M}_\lambda[x]$ and

$$\mathcal{M}_\lambda[x] = \begin{pmatrix} 0 & \frac{2\lambda}{2(\lambda+1)} & & & \\ \frac{1}{2\lambda} & 0 & \frac{2\lambda+1}{2(\lambda+2)} & & \\ & \frac{2}{2(\lambda+1)} & 0 & \frac{2\lambda+2}{2(\lambda+3)} & \\ & & \frac{3}{2(\lambda+2)} & 0 & \ddots \\ & & & \ddots & \ddots \end{pmatrix}. \tag{2.3c}$$

Another way to construct these multiplication operators is by the explicit formula

$$\mathcal{M}_\lambda[a^\lambda]_{j,k} = \sum_{s=\max(0,k-j)}^{k} a_{2s+j-k}^\lambda(k, 2s+j-k), \quad j,k \geq 0, \tag{2.4}$$

where $c_s^\lambda(j,k)$ can be evaluated recursively following the remark in Olver & Townsend (2013, §3.1).

Our solution to the nonlinear ODEs relies on a third way to express $\mathcal{M}_\lambda[a^\lambda]$, which allows fast applications of truncations of $\mathcal{M}_\lambda[a^\lambda]$ to vectors. See Section 3 below.

When $\mathcal{D}_\lambda$ and $\mathcal{M}_\lambda[a^\lambda]$ are employed, each term in (2.2) maps to a different ultraspherical basis. So the following conversion operators $\mathcal{S}_0$ and $\mathcal{S}_\lambda$ are used to map the coefficients in Chebyshev T and $C^{(\lambda)}$

to those in $C^{(1)}$ and $C^{(\lambda+1)}$, respectively

$$\mathcal{S}_0 = \begin{pmatrix} 1 & -\frac{1}{2} & & & \\ & \frac{1}{2} & -\frac{1}{2} & & \\ & & \frac{1}{2} & -\frac{1}{2} & \\ & & & \ddots & \ddots \end{pmatrix} \quad \text{and} \quad \mathcal{S}_\lambda = \begin{pmatrix} 1 & & -\frac{\lambda}{\lambda+2} & & \\ & \frac{\lambda}{\lambda+1} & & -\frac{\lambda}{\lambda+3} & \\ & & \frac{\lambda}{\lambda+2} & & -\frac{\lambda}{\lambda+4} \\ & & & \ddots & \end{pmatrix}$$

for $\lambda \geq 1$. In terms of these operators, the differential operator (2.2) is represented as

$$\mathcal{L} = \mathcal{M}_N[a^N]\mathcal{D}_N + \sum_{\lambda=0}^{N-1} \mathcal{S}_{N-1} \dots \mathcal{S}_\lambda \mathcal{M}_\lambda[a^\lambda]\mathcal{D}_\lambda, \tag{2.5}$$

and (2.1) becomes

$$\mathcal{L}u = \mathcal{S}_{N-1} \dots \mathcal{S}_0 f, \tag{2.6}$$

where both the sides are coefficients in $C^{(N)}$.

If $a^\lambda(x)$ is analytic or many-times differentiable, it can often be approximated to machine precision by an ultraspherical approximant of low degree, resulting in the infinite multiplication matrices $\mathcal{M}_0[a^0]$ and $\mathcal{M}_\lambda[a^\lambda]$ being banded. Thus, $a^\lambda(x)$ is assumed to be an ultraspherical series, and, if the degree of $a^\lambda(x)$ is denoted by d^λ, the bandwidths of $\mathcal{M}_\lambda[a^\lambda]$ are both d^λ. Therefore, the λth-order operator in the sum in (2.5) has the upper and lower bandwidths $d^\lambda + 2N - \lambda$ and $d^\lambda - \lambda$, respectively.

To obtain a system of finite dimension, one truncates $\mathcal{L}$ by premultiplying $\mathcal{P}_{n-N}$ and postmultiplying $\mathcal{P}_n^\top$, where the projection operator $\mathcal{P}_n = (I_n, \mathbf{0})$. Incorporating the first n columns of the boundary conditions gives us an $n \times n$ square system

$$\begin{pmatrix} \mathcal{B}\mathcal{P}_n^\top \\ \mathcal{P}_{n-N}\mathcal{L}\mathcal{P}_n^\top \end{pmatrix} \mathcal{P}_n \boldsymbol{u} = \begin{pmatrix} c \\ \mathcal{P}_{n-N}\mathcal{S}_{N-1} \dots \mathcal{S}_0 \mathcal{P}_n^\top \mathcal{P}_n \boldsymbol{f} \end{pmatrix}, \tag{2.7}$$

where the unknown $\mathcal{P}_n\boldsymbol{u}$ and the (unconverted) right-hand side $\mathcal{P}_n\boldsymbol{f}$ are n-vectors. Solving (2.7) gives the Chebyshev coefficients of the n-truncation of the solution $\tilde{u}_n(x) = \sum_{j=0}^{n-1} u_j T_j(x)$.

The matrix on the left-hand side of (2.7) is almost-banded with the upper and lower bandwidths both $m = N + \max_\lambda(d^\lambda - \lambda)$, when $n > m$.

The ultraspherical spectral method recapitulated above enjoys three key advantages over the collocation-based pseudospectral methods:

- Since (2.7) is an almost-banded system, it can be solved in $\mathcal{O}(m^2 n)$ flops, where n and m are the degrees of freedom of the solution vector and the bandwidth, respectively.
- The condition number of (2.7) is effectively constant (Olver & Townsend, 2013; Cheng & Xu, 2024).
- The forward error of the computed solution can be read directly from the right-hand side of the linear system in the course of solving (2.7) by QR factorization. This helps determine the minimal

degrees of freedom that is required to resolve the solution for a preset accuracy tolerance, therefore allowing for adaptivity at virtually no extra cost.[2]

These advantages can be largely retained in solving nonlinear ODEs as we shall see.

2.2 *Linearization, truncation and adaptivity*

In kth iteration of Newton's method, we solve the linearized problem

$$\mathcal{J}[u^k]\delta^k(x) = -\mathcal{F}(u^k)$$

to obtain the update $\delta^k(x)$ for the current iterate $u^k(x)$. $\mathcal{J}[u^k]$, the Fréchet derivative of $\mathcal{F}$ at $u^k(x)$, is a linear differential operator

$$\mathcal{J}[u^k] = a^N(x)\frac{\mathrm{d}^N}{\mathrm{d}x^N} + \ldots + a^1(x)\frac{\mathrm{d}}{\mathrm{d}x} + a^0(x), \tag{2.8}$$

where we slightly abuse the notations by recycling the symbols used in (2.2). Here, we use brackets instead of parentheses in $\mathcal{J}[u^k]$ to emphasise that $\mathcal{J}[u^k]$ is constructed out of $u^k(x)$ in the sense that the variable coefficients $a^\lambda(x)$ depend on $u^k(x)$. It should not be understood as $\mathcal{J}$ acting on $u^k(x)$, as it is $\delta^k(x)$ that $\mathcal{J}[u^k]$ acts on. A standard text on Newton's method in functional space and the Fréchet derivative is, e.g., Atkinson & Han (2005, §5). For the calculation of the Fréchet derivatives by algorithmic differentiation, see, e.g., Griewank & Walther (2008); Naumann (2011); Birkisson & Driscoll (2012).

Besides the right-hand side $\mathcal{F}(u^k)$, the dependence of $a^\lambda(x)$ on $u^k(x)$ is also often in the form of composition of $u^k(x)$. For example, for the nonlinear operator on the left-hand side of the Bratu equation $u'' + \beta e^u = 0$ with a given β, the Fréchet derivative $\mathcal{J}[u] = \frac{\mathrm{d}^2}{\mathrm{d}x^2} + \beta e^u$ at a given u. Here, $a^0(x) = \beta e^{u(x)}$ is a scaled composition of the exponential function and $u(x)$. Multiple approaches are available to calculate such a composition. The simplest one is to sample the composition, e.g., $e^{u^k(x)}$, on Chebyshev grids of increasing sizes and calculate the Chebyshev coefficients by discrete cosine transform or fast Fourier transform (FFT) until the composition is fully resolved. With the variable coefficients $a^\lambda(x)$ available in its Chebyshev or ultraspherical coefficients, we have

$$\mathcal{J}[u^k] = \mathcal{M}_N[a^N]\mathcal{D}_N + \sum_{\lambda=0}^{N-1} \mathcal{S}_{N-1}\ldots\mathcal{S}_\lambda\mathcal{M}_\lambda[a^\lambda]\mathcal{D}_\lambda. \tag{2.9}$$

Analogously, linearization gives

$$\mathcal{N}'[u^k]\delta^k(x) = -\mathcal{N}(u^k), \tag{2.10}$$

where $\mathcal{N}'[u^k]$, the Fréchet derivative of the boundary condition $\mathcal{N}$ at $u^k(x)$, has dimension $N \times \infty$.

[2] This is exactly how adaptivity is effected in ApproxFun (Olver, 2019).

We truncate (2.9) and (2.10) to have

$$J_n^k \delta^k = f^k, \tag{2.11}$$

where $\delta^k = (\delta_0^k, \delta_1^k, \ldots, \delta_{n-1}^k)^T$,

$$J_n^k = \begin{pmatrix} \mathcal{N}'\mathcal{P}_n^\top \\ \mathcal{P}_{n-N}\mathcal{J}[u^k]\mathcal{P}_n^\top \end{pmatrix} \text{ and } f^k = -\begin{pmatrix} \mathcal{N}(u^k) \\ \mathcal{P}_{n-N}\mathcal{S}_{N-1}\cdots\mathcal{S}_0\mathcal{P}_n^\top\mathcal{P}_n\mathcal{F}(u^k) \end{pmatrix}.$$

The subscript n in J_n^k is used to indicate the dimension of J_n^k.

The dependence of $a^\lambda(x)$ on $u^k(x)$ suggests that the bandwidth of $\mathcal{M}_\lambda[a^\lambda]$ becomes proportional to its size, therefore implying the same proportionality between the bandwidth and the dimension of J_n^k. In many cases, the bandwidth and the dimension of J_n^k could be roughly equal, i.e., $m \approx n$, rendering the loss of the bandedness. For example, for the Bratu equation $m = n$, implying a dense system with no trace of bandedness at all.

If (2.11) were to be solved by the QR factorization as in Olver & Townsend (2013), adaptivity would be effected as in the linear case, despite of the loss of bandedness. However, since we choose to solve (2.11) inexactly using GMRES (see below), adaptivity has to be realized in another way. The strategy we follow is the one introduced by Aurentz & Trefethen (2017). Specifically, the dimension of (2.11) is initially determined by the degrees of $a^\lambda(x)$ for all λ and the degree of the residual $\mathcal{F}(u^k)$. That is, we choose

$$n = \max\left(N + \max_\lambda(d^\lambda - \lambda), d_\mathcal{F}\right) + 1, \tag{2.12}$$

where d^λ is reused to denote the degree of $a^\lambda(x)$ in (2.9) and $d_\mathcal{F}$ is the degree of $\mathcal{F}(u^k)$. Since we are essentially working with functions in the Newton–Kantorovich framework (Birkisson, 2013), it is natural to require the emergence of a plateau in δ^k before it can be deemed as fully resolved. If such a plateau is not seen, we double the dimensions of the system to deal with the rapid growth of the high-frequency components due to some of the most common nonlinearities, such as u^2. Thus, we solve (2.11) of the initial size (2.12), check the resolution, augment the system and repeat until the solution is eventually satisfactorily resolved. Finally, the solution is 'chopped' to trim off the unnecessary trailing coefficients of small magnitude. Note that each time n is doubled, the bandedness seems restored with the bandwidth being half of the dimension of the system. Despite the bandedness, this proportionality still suggests $\mathcal{O}(n^3)$ operations for solving the system via a direct method.

2.3 *Inexact Newton condition*

Instead of solving (2.11) exactly, we choose to solve it by enforcing only the inexact Newton condition

$$\left\|J_n^k \delta^k - f^k\right\| \leq \omega^k \left\|f^k\right\|, \tag{2.13}$$

where $\omega^k \in [0, 1)$ is the forcing term. The purpose of choosing a nonzero ω^k is to solve (2.11) for δ^k to just enough precision so that good progress can still be made when the current approximation

is far from a solution, but also to obtain quadratic convergence when near a solution (Eisenstat & Walker, 1994, 1996; Kelley, 1995). This immediately suggests the use of Krylov subspace methods, e.g., GMRES, as these methods produce inexact solutions cheaply. That is, once (2.13) is satisfied, the Krylov subspace iteration of GMRES terminates and returns an inexact solution δ^k. The adoption of the inexact Newton condition and the use of GMRES are justified in Sections 3, 4 and 5 where we see how the full potential of the ultraspherical spectral method can be unleashed in the current context, forming an extremely efficient framework for solving nonlinear ODE boundary value problems.

In existing implementations of the ultraspherical spectral method for nonlinear problems (Driscoll *et al.*, 2014; Olver, 2019), (2.11) is usually explicitly formed before solved by direct methods, e.g., LU or QR, as in the linear case, thus enforcing effectively the exact Newton condition.

2.4 *Global Newton methods*

A naive implementation of Newton's method based on (2.11), sometimes regarded as the local Newton method, has limited chance of convergence unless the initial iterate is close enough to the solution. Thus, a practical global Newton method must be applied. The global Newton method for solving nonlinear systems has many variants, and they differ mainly by the strategy for determining the search direction and the step length, how the linear systems are solved and whether the derivative is dispensed with. We use three global Newton methods as the vehicle for implementing the INGU framework—the trust region method (Nocedal & Wright, 2006, §11.2) enabled by the dogleg approximation (TR–dogleg) (Powell, 1970), the line search with Armijo backtracking (LS–backtracking) (Kelley, 1995, §8) and the trust region method in the affine contravariant framework (TR–contravariant) (Deuflhard, 2005, §3.2).

The TR–dogleg method is arguably the most reliable and widely accepted algorithm, being the default implementation in many mainstream computing platforms, standard libraries or public domain codes. However, there is a catch—the TR–dogleg requires the knowledge of the transpose of the Jacobian or the access to it via its products with vectors. The line search method is relatively easy to implement, and this is true particularly for its backtracking-based implementation. The TR–contravariant method assumes that $\mathcal{F}(u)$ satisfies an affine contravariant Lipschitz condition based on which the minimization of the residual is modeled as a constrained quadratic optimization problem. Though it is not as well received as the TR–dogleg and LS–backtracking methods, TR–contravariant performs equally well on average in our experiments.

2.5 *A prototype framework*

We now have the key ingredients for setting up the prototype INGU framework, which is summarized in Algorithm 1. For convenience, we denote by $\mathcal{G}$ the operator formed by concatenating $\mathcal{N}$ and $\mathcal{F}$ vertically as $[\mathcal{N}; \mathcal{F}]$, where MATLAB syntax is used. This way, when $\mathcal{G}$ is applied to a solution u^k, it returns a column vector containing the residual of both the nonlinear equation and the boundary conditions.

There are three nested loops in this framework. As the outer loop (lines 2–9), the Newton iteration generates the sequence of the updates δ^k and the approximate solutions u^k. The initial iterate is usually chosen to be the polynomial of lowest degree that satisfies the boundary conditions. The outer iteration terminates when the residual $\mathcal{G}(u^k)$ is smaller than the preset tolerance. This tolerance usually includes terms for both the relative and absolute residuals.

The intermediate loop (lines 3–6) ensures that the update δ^k has an adequate resolution by repeatedly doubling the size of the linear system until a plateau is formed in δ^k. A zero vector is usually used as the

Algorithm 1 INGU prototype

Inputs: nonlinear operator $\mathcal{G}$, Fréchet operator $\mathcal{J}$, initial iterate u^0, relative tolerance η_r for the residual.
Output: approximate solution u^k satisfies $\left\|\mathcal{G}(u^k)\right\| \leq \eta_r \left\|\mathcal{G}(u^0)\right\| + \eta_r$.

1. Set $k = 0$, construct u^0, $\eta = \eta_r \left\|\mathcal{G}(u^0)\right\| + \eta_r$.
2. **while** $\left\|\mathcal{G}(u^k)\right\| > \eta$ **do** ▷ outer iteration
3. **while** δ^k is not resolved **do** ▷ intermediate iteration
4. Double the size of the system.
5. Solve (2.11) inexactly by GMRES. ▷ inner iteration
6. **end while**
7. Call Postprocess.
8. $k = k + 1$
9. **end while**
10. **return** u^k

initial iterate of the intermediate loop. The solution δ^k from the last intermediate iteration is zero-padded to double its length before being used as the initial iterate for GMRES in the subsequent intermediate iteration. In most of our experiments, the size of the system is doubled only a couple of times before the intermediate loop terminates. See Section 6 for more details.

The Krylov subspace iteration of GMRES is deemed as the inner loop (line 4) and is terminated when (2.13) is satisfied. The forcing term ω^k is determined by the function Postprocess from the previous outer iteration. The function Postprocess usually takes in the information of the current outer iteration, such as the residual and the Jacobian, and the information inherited from the previous outer iteration, such as the contraction factor and the contravariant Kantorovich quantity for the TR–contravariant method. The outputs of Postprocess usually include an updated solution u^{k+1} obtained by adding to u^k a post-processed Newton step $\tilde{\delta}^k$, a new forcing term ω^{k+1} and the size of the trust region for the next outer iteration. See Algorithms 3, 4 and 5 in Appendix A for details of the function Postprocess for each global Newton method.

3. Fast matrix-vector multiplication

The efficiency of the GMRES method hinges on if fast matrix-vector multiplication is available. Specifically, we wish to be able to apply J_n^k to n-vectors speedily, despite the loss of the bandedness in J_n^k.

3.1 *A basic approach*

For the moment, let us ignore the top N rows of J_n^k, i.e., the boundary conditions, and examine $\mathcal{P}_{n-N}\mathcal{J}\mathcal{P}_n^\top$ in a termwise manner. We first look at the zeroth-order term that is a truncation of $\mathcal{S}_{N-1}\dots\mathcal{S}_0\mathcal{M}_0[a^0]$. Denote by $\mathcal{T}[a^0]$, $\mathcal{H}[a^0]$ and $\mathcal{R}[a^0]$ the Toeplitz, the Hankel and the rank-1 parts of

$2\mathcal{M}_0[a^0]$, respectively, i.e., $\mathcal{M}_0[a^0] = \big(\mathcal{T}[a^0] + \mathcal{H}[a^0] + \mathcal{R}[a^0]\big)/2$, where

$$\mathcal{T}[a^0] = \begin{pmatrix} 2a_0 & a_1 & a_2 & a_3 & \cdots \\ a_1 & 2a_0 & a_1 & a_2 & \ddots \\ a_2 & a_1 & 2a_0 & a_1 & \ddots \\ a_3 & a_2 & a_1 & 2a_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}, \quad \mathcal{H}[a^0] = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & \cdots \\ a_1 & a_2 & a_3 & a_4 & \iddots \\ a_2 & a_3 & a_4 & a_5 & \iddots \\ a_3 & a_4 & a_5 & a_6 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix},$$

and $\mathcal{R}[a^0] = -\boldsymbol{e}_1\boldsymbol{a}^0$. Here, $\boldsymbol{e}_1$ is the first column of $\mathcal{I}$, and $\boldsymbol{a}^0$ is the infinite vector obtained by prolonging $a^0 = (a_0, a_1, \ldots, a_{d^0})$ with zeros. For an n-vector v, calculating each of $s_T = \mathcal{P}_n\mathcal{T}[a^0]\mathcal{P}_n^\top v$ and $s_H = \mathcal{P}_n\mathcal{H}[a^0]\mathcal{P}_n^\top v$ costs two FFTs and one inverse FFT, all of length $2n$ (Golub & Van Loan, 2013, P4.8.6). Calculating $s_R = \mathcal{P}_n\mathcal{R}[a^0]\mathcal{P}_n^\top v$, $s = s_T + s_H + s_R$ and $\mathcal{P}_{n-N}\mathcal{S}_{N-1}\ldots\mathcal{S}_0\mathcal{P}_n^\top s$ can be done in $\mathcal{O}(n)$ flops. Hence, the total cost of applying $\mathcal{P}_{n-N}\mathcal{S}_{N-1}\ldots\mathcal{S}_0\mathcal{M}_0[a^0]\mathcal{P}_n^\top$ is dominated by the 6 FFTs of length $2n$.

The first-order term $\mathcal{P}_{n-N}\mathcal{S}_{N-1}\ldots\mathcal{S}_1\mathcal{M}_1[a^1]\mathcal{D}_1\mathcal{P}_n^\top$ can also be quickly applied, as pointed out by Olver & Townsend (2013, Remark 3). To see this, we observe that $\mathcal{M}_1[a^1]$ is constructed using $a^1(x)$'s $C^{(1)}$ coefficients $a^1 = (a_0^1, a_1^1, \ldots, a_{d^1}^1)$ and it also acts on and maps to coefficient vectors in $C^{(1)}$. This suggests another way to express it

$$\mathcal{M}_1[a^1] = \mathcal{S}_0\mathcal{M}_0\big[\mathcal{P}_{d^1+1}\mathcal{S}_0^{-1}\mathcal{P}_{d^1+1}^\top a^1\big]\mathcal{S}_0^{-1},$$

where $\mathcal{M}_0[\mathcal{P}_{d^1+1}\mathcal{S}_0^{-1}\mathcal{P}_{d^1+1}^\top a^1]$ means to first convert the $C^{(1)}$ coefficients of $a^1(x)$ to the Chebyshev ones and then construct the $\mathcal{M}_0$ multiplication operator with the Chebyshev coefficients of $a^1(x)$. In practice, $a^1(x)$ is often available by its Chebyshev coefficients, either as the variable coefficient of a linear term in $\mathcal{F}(u)$ or a composition of $u^k(x)$ or a combination of both. Hence, we assume $\hat{a}^1 = \mathcal{P}_{d^1+1}\mathcal{S}_0^{-1}\mathcal{P}_{d^1+1}^\top a^1 = (\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_{d^1})$ is available from now on and replace $\mathcal{M}_1[a^1]$ by $\mathcal{M}_1[\hat{a}^1]$ when it is constructed using $\hat{a}^1$.

By definition, the displacement of $\mathcal{M}_0[\hat{a}^1]$, also known as Sylvester map, is

$$\nabla_{\mathcal{S}_0}\big(\mathcal{M}_0[\hat{a}^1]\big) := \mathcal{S}_0\mathcal{M}_0[\hat{a}^1] - \mathcal{M}_0[\hat{a}^1]\mathcal{S}_0, \tag{3.1}$$

whose linearity leads to

$$\nabla_{\mathcal{S}_0}\big(\mathcal{M}_0[\hat{a}^1]\big) = \frac{1}{2}\big(\nabla_{\mathcal{S}_0}\big(\mathcal{T}[\hat{a}^1]\big) + \nabla_{\mathcal{S}_0}\big(\mathcal{H}[\hat{a}^1]\big) + \nabla_{\mathcal{S}_0}\big(\mathcal{R}[\hat{a}^1]\big)\big),$$

where $\mathcal{T}[\hat{a}^1]$, $\mathcal{H}[\hat{a}^1]$ and $\mathcal{R}[\hat{a}^1]$ are the Toeplitz, the Hankel and the rank-1 parts of $\mathcal{M}_0[\hat{a}^1]$, respectively. Some algebraic work then gives

$$\nabla_{\mathcal{S}_0}\big(\mathcal{T}[\hat{a}^1]\big) = \boldsymbol{c}_t\boldsymbol{r}_{th}, \quad \nabla_{\mathcal{S}_0}\big(\mathcal{H}[\hat{a}^1]\big) = \hat{\mathcal{H}}[\hat{a}^1] + \boldsymbol{c}_h\boldsymbol{r}_{th}, \quad \nabla_{\mathcal{S}_0}\big(\mathcal{R}[\hat{a}^1]\big) = \boldsymbol{c}_r\boldsymbol{r}_r, \tag{3.2}$$

where

$$\boldsymbol{c}_t = \frac{1}{2}\begin{pmatrix} 2 & 0 & 0 & 0 & \cdots \\ \hat{a}_0+\hat{a}_2 & \hat{a}_1+\hat{a}_3 & \hat{a}_2+\hat{a}_4 & \hat{a}_3+\hat{a}_5 & \cdots \\ \hat{a}_1 & \hat{a}_2 & \hat{a}_3 & \hat{a}_4 & \cdots \end{pmatrix}^\top, \quad \boldsymbol{r}_{th} = \frac{1}{2}\begin{pmatrix} \hat{a}_0 & \hat{a}_1 & \hat{a}_2 & \hat{a}_3 & \cdots \\ -2 & 0 & 0 & 0 & \cdots \\ 0 & -2 & 0 & 0 & \cdots \end{pmatrix},$$

$$\hat{\mathcal{H}}[\hat{a}^1] = \frac{1}{2}\begin{pmatrix} \hat{a}_{|-2|}-\hat{a}_2 & \hat{a}_{|-1|}-\hat{a}_3 & \hat{a}_0-\hat{a}_4 & \hat{a}_1-\hat{a}_5 & \cdots \\ \hat{a}_{|-1|}-\hat{a}_3 & \hat{a}_0-\hat{a}_4 & \hat{a}_1-\hat{a}_5 & \hat{a}_2-\hat{a}_6 & \iddots \\ \hat{a}_0-\hat{a}_4 & \hat{a}_1-\hat{a}_5 & \hat{a}_2-\hat{a}_6 & \hat{a}_3-\hat{a}_7 & \iddots \\ \hat{a}_1-\hat{a}_5 & \hat{a}_2-\hat{a}_6 & \hat{a}_3-\hat{a}_7 & \hat{a}_4-\hat{a}_8 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix},$$

$$\boldsymbol{c}_h = \frac{1}{2}\begin{pmatrix} 2 & 0 & 0 & 0 & \cdots \\ \hat{a}_{|-2|}+\hat{a}_0 & \hat{a}_{|-1|}+\hat{a}_1 & \hat{a}_0+\hat{a}_2 & \hat{a}_1+\hat{a}_3 & \cdots \\ \hat{a}_{|-1|} & \hat{a}_0 & \hat{a}_1 & \hat{a}_2 & \cdots \end{pmatrix}^\top,$$

$$\boldsymbol{c}_r = -\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \end{pmatrix}^\top, \quad \boldsymbol{r}_r = \frac{1}{2}\begin{pmatrix} 0 & \hat{a}_1 & \hat{a}_0+\hat{a}_2 & \hat{a}_1+\hat{a}_3 & \cdots \end{pmatrix}.$$

Here, $\hat{\mathcal{H}}[\hat{a}^1]$ is, again, a Hankel operator. Absolute values are used in some of the subscripts in $\boldsymbol{c}_h$ and $\hat{\mathcal{H}}[\hat{a}^1]$ to reveal the pattern. Equation (3.2) shows that the displacement $\nabla_{\mathcal{S}_0}\left(\mathcal{M}_0[\hat{a}^1]\right)$ is a Hankel-plus-low-rank operator. Using (3.1) to express $\mathcal{M}_1[\hat{a}^1]$ in terms of $\nabla_{\mathcal{S}_0}\left(\mathcal{M}_0[\hat{a}^1]\right)$, we have

$$\begin{aligned}\mathcal{M}_1[\hat{a}^1] &= \nabla_{\mathcal{S}_0}\big(\mathcal{M}_0[\hat{a}^1]\big)\mathcal{S}_0^{-1} + \mathcal{M}_0[\hat{a}^1] \\ &= \frac{1}{2}\big(\big(\hat{\mathcal{H}}[\hat{a}^1] + (\boldsymbol{c}_t+\boldsymbol{c}_h)\boldsymbol{r}_{th} + \boldsymbol{c}_r\boldsymbol{r}_r\big)\mathcal{S}_0^{-1} + \mathcal{T}[\hat{a}^1] + \mathcal{H}[\hat{a}^1] + \mathcal{R}[\hat{a}^1]\big). \end{aligned} \tag{3.3}$$

Using MATLAB's notation `triu` to denote the upper triangular part of an operator, we have

$$\mathcal{S}_0^{-1} = \texttt{triu}\big(\boldsymbol{c}^0\boldsymbol{r}\big), \quad \boldsymbol{c}^0 = \begin{pmatrix} 1 & 0 & 2 & 0 & 2 & \cdots \\ 0 & 2 & 0 & 2 & 0 & \cdots \end{pmatrix}^\top, \quad \boldsymbol{r} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 1 & 0 & 1 & 0 & \cdots \end{pmatrix}. \tag{3.4}$$

With (3.4), the first term in the outermost parentheses of (3.3) can be simplified as

$$\big(\hat{\mathcal{H}}[\hat{a}^1] + (\boldsymbol{c}_t+\boldsymbol{c}_h)\boldsymbol{r}_{th} + \boldsymbol{c}_r\boldsymbol{r}_r\big)\mathcal{S}_0^{-1} = -\begin{pmatrix} \hat{a}_2 & \hat{a}_3 & \hat{a}_4 & \hat{a}_5 & \cdots \\ \hat{a}_3 & \hat{a}_4 & \hat{a}_5 & \hat{a}_6 & \iddots \\ \hat{a}_4 & \hat{a}_5 & \hat{a}_6 & \hat{a}_7 & \iddots \\ \hat{a}_5 & \hat{a}_6 & \hat{a}_7 & \hat{a}_8 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix} - \mathcal{H}[\hat{a}^1] - \mathcal{R}[\hat{a}^1].$$

The last equation and (3.3) show that $\mathcal{M}_1[\hat{a}^1]$ is a Toeplitz-plus-Hankel operator given by

$$\mathcal{M}_1[\hat{a}^1] = \frac{1}{2}\left[\begin{pmatrix} 2\hat{a}_0 & \hat{a}_1 & \hat{a}_2 & \hat{a}_3 & \cdots \\ \hat{a}_1 & 2\hat{a}_0 & \hat{a}_1 & \hat{a}_2 & \ddots \\ \hat{a}_2 & \hat{a}_1 & 2\hat{a}_0 & \hat{a}_1 & \ddots \\ \hat{a}_3 & \hat{a}_2 & \hat{a}_1 & 2\hat{a}_0 & \ddots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} - \begin{pmatrix} \hat{a}_2 & \hat{a}_3 & \hat{a}_4 & \hat{a}_5 & \cdots \\ \hat{a}_3 & \hat{a}_4 & \hat{a}_5 & \hat{a}_6 & \iddots \\ \hat{a}_4 & \hat{a}_5 & \hat{a}_6 & \hat{a}_7 & \iddots \\ \hat{a}_5 & \hat{a}_6 & \hat{a}_7 & \hat{a}_8 & \iddots \\ \vdots & \iddots & \iddots & \iddots & \iddots \end{pmatrix}\right].$$

Hence, when the Chebyshev coefficients of $a^1(x)$ are available, the cost of multiplying $\mathcal{P}_{n-N}\mathcal{S}_{N-1}\dots\mathcal{S}_1\mathcal{M}_1[a^1]\mathcal{D}_1\mathcal{P}_n^\top$ with a vector is again 6 FFTs of length $2n$ plus $\mathcal{O}(n)$ flops.

We wish the higher order terms in $\mathcal{J}[u^k]$ are structured alike so that fast multiplications can be effected similarly. Unfortunately, whether general higher order multiplication operators bear a Toeplitz plus Hankel form, or something akin to to allow a fast application is not known. Fortunately, we can always circumvent this via $\mathcal{M}_1[\hat{a}^1]$. Keeping in mind that $\mathcal{M}_\lambda[a^\lambda]$ is constructed by using $a^\lambda(x)$'s $C^{(\lambda)}$ coefficients $a^\lambda = (a_0^\lambda, a_1^\lambda, \dots, a_{d^\lambda}^\lambda)$ and that it maps between $C^{(\lambda)}$ coefficients, we re-express it as

$$\mathcal{M}_\lambda[a^\lambda] = \mathcal{S}_{\lambda-1}\dots\mathcal{S}_1\mathcal{M}_1[\hat{a}^\lambda]\mathcal{S}_1^{-1}\dots\mathcal{S}_{\lambda-1}^{-1}, \text{ for } \lambda \geq 2, \tag{3.5}$$

where $\hat{a}^\lambda = \mathcal{P}_{d^\lambda+1}\mathcal{S}_0^{-1}\dots\mathcal{S}_{\lambda-1}^{-1}\mathcal{P}_{d^\lambda+1}^\top a^\lambda$ are the Chebyshev coefficients of $a^\lambda(x)$ and $\mathcal{S}_\lambda^{-1}$ can be expressed explicitly as

$$\mathcal{S}_\lambda^{-1} = \mathtt{triu}\big(\boldsymbol{c}^\lambda \boldsymbol{r}\big), \quad \boldsymbol{c}^\lambda = \frac{1}{\lambda}\begin{pmatrix} \lambda & 0 & \lambda+2 & 0 & \lambda+4 & \cdots \\ 0 & \lambda+1 & 0 & \lambda+3 & 0 & \cdots \end{pmatrix}^\top, \text{ for } \lambda \geq 1.$$

Since $\mathcal{S}_\lambda^{-1}$ is the upper triangular part of a rank-2 operator, applying $\mathcal{S}_\lambda^{-1}$ to an n-vector has a complexity of only $\mathcal{O}(n)$ flops.

Finally, substituting (3.5) into (2.9) yields

$$\mathcal{J}[u^k] = \mathcal{S}_{N-1}\dots\mathcal{S}_1\left(\sum_{\lambda=1}^{N}\mathcal{M}_1[\hat{a}^\lambda]\mathcal{S}_1^{-1}\dots\mathcal{S}_{\lambda-1}^{-1}\mathcal{D}_\lambda + \mathcal{S}_0\mathcal{M}_0[a^0]\right), \tag{3.6}$$

which shows that the multiplication part of each term in $\mathcal{J}[u^k]$ can be done via $\mathcal{M}_1$ or $\mathcal{M}_0$. Since multiplying any of $\mathcal{S}_0, \dots, \mathcal{S}_{N-1}, \mathcal{S}_1^{-1}, \dots, \mathcal{S}_{N-1}^{-1}, \mathcal{D}_1, \dots, \mathcal{D}_N$ with an n-vector costs $\mathcal{O}(n)$, the cost of applying $\mathcal{P}_{n-N}\mathcal{J}[u^k]\mathcal{P}_n^\top$ is mainly the $6(N+1)$ FFTs of length $2n$.

Standard FFT libraries, like FFTW (Frigo & Johnson, 2005), allow the users to pre-plan an optimized FFT of a given size and apply the plan repeatedly to vectors of the same size. This can help accelerate each of the intermediate iteration for the dimension of the system is unchanged throughout.

Noting that applying the boundary rows $\mathcal{N}'\mathcal{P}_n^\top$ to v can be done in $\mathcal{O}(Nn)$ flops, we conclude that the multiplication of J_n^k and v costs only $\mathcal{O}(Nn\log_2 n)$ flops. This justifies the use of GMRES.

We also remark that computing f^k costs at most $\mathcal{O}(n\log_2 n)$ flops, since $\mathcal{F}(u^k)$ is a composition of u^k and $\mathcal{N}(u^k)$ are N functionals.

3.2 *Even fewer FFTs*

The cost of $6(N+1)$ FFTs can be further reduced. Noting that $\mathcal{M}_1[a^0]\mathcal{S}_0 = \mathcal{S}_0\mathcal{M}_0[a^0]$, we can rewrite (3.6) as

$$\mathcal{J}[u^k] = \mathcal{S}_{N-1}\dots\mathcal{S}_1\left(\sum_{\lambda=1}^{N}\mathcal{M}_1[\hat{a}^\lambda]\mathcal{S}_1^{-1}\dots\mathcal{S}_{\lambda-1}^{-1}\mathcal{D}_\lambda + \mathcal{M}_1[\hat{a}^0]\mathcal{S}_0\right), \tag{3.7}$$

where we let $\hat{a}^0 = a^0$ to facilitate the notation below. This way, the inverse FFT for each of the $\mathcal{M}_1$ terms can be factored out of the parentheses. To see this, let

$$\mathcal{P}_n\mathcal{M}_1[\hat{a}^\lambda]\mathcal{P}_n^T = T^\lambda + H^\lambda,$$

where T^λ and H^λ are the Toeplitz and the Hankel parts, respectively. Let w be an n-vector. To compute $T^\lambda w$ by FFTs, we embed T^λ into a $2n \times 2n$ circulant matrix by choosing $A^\lambda \in \mathbb{C}^{n\times n}$ carefully, i.e.,

$$\begin{pmatrix} T^\lambda w \\ \times \end{pmatrix} = \begin{pmatrix} T^\lambda & A^\lambda \\ A^\lambda & T^\lambda \end{pmatrix}\begin{pmatrix} w \\ 0 \end{pmatrix} = F_{2n}^{-1}\mathtt{diag}\left(F_{2n}t^\lambda\right)F_{2n}\begin{pmatrix} w \\ 0 \end{pmatrix},$$

where $t^\lambda \in \mathbb{C}^{2n}$ denotes the first column of the circulant matrix, and $F_{2n} \in \mathbb{C}^{2n\times 2n}$ and $F_{2n}^{-1} \in \mathbb{C}^{2n\times 2n}$ are the forward and inverse DFT matrices that we effect by FFT and inverse FFT, respectively. Analogously, for the Hankel part

$$\begin{pmatrix} H^\lambda w \\ \times \end{pmatrix} = \begin{pmatrix} \mathtt{fliplr}(H^\lambda) & B^\lambda \\ B^\lambda & \mathtt{fliplr}(H^\lambda) \end{pmatrix}\begin{pmatrix} \mathtt{flipud}(w) \\ 0 \end{pmatrix} = F_{2n}^{-1}\mathtt{diag}\left(F_{2n}h^\lambda\right)F_{2n}\begin{pmatrix} \mathtt{flipud}(w) \\ 0 \end{pmatrix},$$

where $h^\lambda \in \mathbb{C}^{2n}$ is again the first column of the circulant matrix formed by $\mathtt{fliplr}(H^\lambda)$ and a carefully chosen B^λ. Since all T^λ and H^λ share the same F_{2n}^{-1}, the inverse DFT matrix can be effected by one inverse FFT for all $\mathcal{M}_1[\hat{a}^\lambda]$. See Algorithm 2 for details and the stepwise costs.

Hence, besides the construction of $\hat{a}^\lambda$, the major cost of applying $\mathcal{J}[u^k]$ to an n-vector is about $4(N+1)+1$ FFTs of length $2n$. Since it is common that not every term in $\mathcal{F}(u)$ is nonlinear or premultiplied by a variable coefficient and no FFT is involved for terms with a constant coefficient, this is likely to be an overestimate for many problems.

3.3 *Exact truncation*

So far, we have been slack in truncating the operators and used only square truncations for simplicity. Rigorously, it follows that the exact truncation of $\mathcal{J}[u^k]$ in (3.7) should be calculated as

$$\begin{aligned}\mathcal{P}_{n-N}\mathcal{J}[u^k]\mathcal{P}_n^\top = {} & \left(\mathcal{P}_{n-N}\mathcal{S}_{N-1}\mathcal{P}_{n-N+2}^\top\right)\dots\left(\mathcal{P}_{n+N-4}\mathcal{S}_1\mathcal{P}_{n+N-2}^\top\right)\left[\sum_{\lambda=1}^{N}\left(\mathcal{P}_{n+N-2}\mathcal{M}_1[\hat{a}^\lambda]\mathcal{P}_{n-\lambda}^\top\right)\right. \\ & \left.\times\left(\mathcal{P}_{n-\lambda}\mathcal{S}_1^{-1}\mathcal{P}_{n-\lambda}^\top\right)\dots\left(\mathcal{P}_{n-\lambda}\mathcal{S}_{\lambda-1}^{-1}\mathcal{P}_{n-\lambda}^\top\right)\left(\mathcal{P}_{n-\lambda}\mathcal{D}_\lambda\mathcal{P}_n^\top\right) + \left(\mathcal{P}_{n+N-2}\mathcal{M}_1[\hat{a}^0]\mathcal{P}_n^\top\right)\left(\mathcal{P}_n\mathcal{S}_0\mathcal{P}_n^\top\right)\right].\end{aligned} \tag{3.8}$$

Algorithm 2 Fast Jacobian-vector multiplication

Inputs: A Fréchet operator $\mathcal{J}[u^k]$ in the form of (2.9), the order N of $\mathcal{J}[u^k]$ and an n-vector v.
Output: An $(n-N)$-vector $y = \mathcal{P}_{n-N}\mathcal{J}[u^k]\mathcal{P}_n^\top v$.

1. Calculate the Chebyshev coefficients of $\hat{a}^\lambda$ for all λ. ▷ $\mathcal{O}(n\log_2 n)$
2. Calculate $w^0 = \mathcal{P}_n\mathcal{S}_0\mathcal{P}_n^\top v$ ▷ $\mathcal{O}(n)$
3. and $w_s = \texttt{diag}(F_{2n}t^0)F_{2n}\begin{pmatrix} w^0 \\ 0 \end{pmatrix} + \texttt{diag}(F_{2n}h^0)F_{2n}\begin{pmatrix} \texttt{flipud}(w^0) \\ 0 \end{pmatrix}$. ▷ 4 FFT($2n$)s
4. **for** $\lambda = 1$ to N **do**
5. Calculate $w^\lambda = \mathcal{P}_n\mathcal{S}_1^{-1}\ldots\mathcal{S}_{\lambda-1}^{-1}\mathcal{D}_\lambda\mathcal{P}_n^\top v$ ▷ $\mathcal{O}(n)$
6. and $w_s = w_s + \texttt{diag}(F_{2n}t^\lambda)F_{2n}\begin{pmatrix} w^\lambda \\ 0 \end{pmatrix} + \texttt{diag}(F_{2n}h^\lambda)F_{2n}\begin{pmatrix} \texttt{flipud}(w^\lambda) \\ 0 \end{pmatrix}$. ▷ 4 FFT($2n$)s
7. **end for**
8. Calculate $v_s = F_{2n}^{-1}w_s$ ▷ 1 FFT($2n$)
9. Return $y = \mathcal{P}_{n-N}\mathcal{S}_{N-1}\ldots\mathcal{S}_1\mathcal{P}_n^\top v_s(1:n)$. ▷ $\mathcal{O}(n)$

To effect all $\mathcal{M}_1$ operators via FFTs as discussed above, we have to choose the largest common dimension for all $\mathcal{M}_1$, which is determined by $\mathcal{P}_{n+N-2}$ and $\mathcal{P}_{n-\lambda}^\top$ for $\lambda = 0, 1, \ldots, N$. Therefore, the dimension of all $\mathcal{M}_1$ should be $\max_{\lambda=0,\ldots,N}\{n+N-2, n-\lambda\}$, i.e., n when $N = 1$ or $n+N-2$ for $N > 1$. The subscripts of the truncation operators in (3.8) can then be determined accordingly. Our experiments show that the simple square truncations for all the operators do produce virtually identical results for all the experiments.

4. Preconditioner

Our GMRES-based approach is also justified by a simple but effective preconditioner. The fact that $\mathcal{J}[u^k]$ is dense motivates us to use an almost-banded preconditioner—if a diagonal scaling or Jacobi-type preconditioner works perfectly for an almost-banded system, as suggested in Olver & Townsend (2013), why not use an almost-banded one to precondition the dense system where the latter can be deemed as obtained from the former by extending the bandwidth of the former? Hence, we propose the use of a right preconditioner for the kth outer iteration

$$\mathcal{W}^k = \begin{pmatrix} \mathcal{N}' \\ \tilde{\mathcal{J}}[u^k] \end{pmatrix},$$

where

$$\tilde{\mathcal{J}}[u^k] = \mathcal{M}_N[\tilde{a}^N]\mathcal{D}_N + \sum_{\lambda=0}^{N-1}\mathcal{S}_{N-1}\ldots\mathcal{S}_\lambda\mathcal{M}_\lambda[\tilde{a}^\lambda]\mathcal{D}_\lambda. \tag{4.1}$$

Here, the multiplication operators are constructed from the first $m^\lambda + 1$ leading coefficients of a^λ, that is, $\tilde{a}^\lambda = \left(a_0^\lambda, a_1^\lambda, \ldots, a_{m^\lambda}^\lambda\right)$, where $m^\lambda = p + \lambda \ll n$ and the value of integer p is to be determined. Each of the summands $\mathcal{M}_N[\tilde{a}^N]\mathcal{D}_N$ and $\mathcal{S}_{N-1}\ldots\mathcal{S}_\lambda\mathcal{M}_\lambda[\tilde{a}^\lambda]\mathcal{D}_\lambda$ in (4.1) and, therefore, $\tilde{\mathcal{J}}[u^k]$ have the upper

and lower bandwidths $p+2N$ and p, respectively, due to the argument given below (2.6). Hence, $\tilde{\mathcal{J}}[u^k]$ is a banded approximation of $\mathcal{J}[u^k]$, and $\tilde{a}^\lambda$ has its contribution in all the nonzero diagonal entries of $\tilde{\mathcal{J}}[u^k]$. Note that an entry in the band of $\tilde{\mathcal{J}}[u^k]$ differs from the entry in the same position in $\mathcal{J}[u^k]$ as the latter has contribution from every element of a^λ, not just the first $m^\lambda+1$ ones. Approximation to $\mathcal{J}[u^k]$ by banded matrices is also investigated in Huang & Boyd (2016), where the authors recommend the use of unsymmetrical truncations in the upper and the lower bands, particularly for ODEs obtained as the linearizations of the nonlinear ones. However, their analysis is based on an oversimplified model and is limited to a specific second-order ODE, therefore not applicable to the general case that we consider here.

Instead of (4.1), we find it easiest to follow (3.6) or (3.7) to construct the banded part of $\mathcal{W}^k$. Since (4.1) is banded, we construct the (truncated) operators explicitly without using FFTs. Suppose that we use (3.6), that is,

$$\check{\mathcal{J}}[u^k] = \mathcal{S}_{N-1}\ldots\mathcal{S}_1\left(\sum_{\lambda=1}^{N}\mathcal{M}_1[\check{a}^\lambda]\mathcal{S}_1^{-1}\ldots\mathcal{S}_{\lambda-1}^{-1}\mathcal{D}_\lambda+\mathcal{S}_0\mathcal{M}_0[\check{a}^0]\right),$$

where $\check{a}^\lambda = \mathcal{P}_{m^\lambda+1}\mathcal{S}_0^{-1}\ldots\mathcal{S}_{\lambda-1}^{-1}\mathcal{P}_{m^\lambda+1}^\top\tilde{a}^\lambda$ are the Chebyshev coefficients of $\tilde{a}^\lambda(x)=\sum_{j=0}^{m^\lambda}a_j^\lambda C_j^{(\lambda)}(x)$. Because of the bandedness, the equivalence of $\check{\mathcal{J}}[u^k]$ and $\tilde{\mathcal{J}}[u^k]$ can be guaranteed by exact truncations. Instead of solving (2.11), we solve

$$J_n^k\left(W_n^k\right)^{-1}\theta^k = f^k, \tag{4.2}$$

where $W_n^k=\mathcal{P}_n\mathcal{W}^k\mathcal{P}_n^\top$, and δ^k is finally recovered by solving $W_n^k\delta^k=\theta^k$. Note that W_n^k stays unchanged within each inner loop. Thus, it suffices to compute the QR factorization of W_n^k only once for each call of GMRES. Since the bandwidths of W_n^k are both $p+N$, its construction and inversion cost $\mathcal{O}((p+N)n)$ and $\mathcal{O}((p+N)^2n)$ flops, respectively. Thus, we choose

$$p=\left\lfloor\sqrt{\log_2 n}\right\rfloor \tag{4.3}$$

to match up the cost of the FFT-based Jacobian-vector multiplication and the function composition of u^k, resulting in an asymptotic complexity of $\mathcal{O}(n\log_2 n)$. For a nonlinear ODE that is not singularly perturbed, as we shall see in Section 6, p usually has a value below 10. Our experiments suggest that a fixed p of small integral value often works equally well. But (4.3) offers a weak dependence on n and this adaptivity may play a bigger role when we migrate to nonlinear problems in higher spatial dimensions.

Assuming $a^N(x)=1$ in (2.8) and employing virtually the same technique used in the proof of Lemma 4.3 in Olver & Townsend (2013), we can readily show that the right-preconditioned system (4.2) is a compact perturbation of the identity operator in the Banach space ℓ_K^2. For the definition of the norm of ℓ_K^2, see Olver & Townsend (2013, Definition 4.2).

LEMMA 4.1 Assume the boundary operator $\mathcal{N}':\ell_D^2\to\mathbb{C}^N$ is bounded and $a^N(x)=1$. Let $\mathcal{W}^k:\ell_{K+1}^2\to\ell_K^2$ for some $K\in\{D-1,D,\ldots\}$, where D is the smallest integer such that $\mathcal{N}':\ell_D^2\to\mathbb{C}^N$ is bounded. Then

$$\begin{pmatrix}\mathcal{N}'\\ \mathcal{J}[u^k]\end{pmatrix}\left(\mathcal{W}^k\right)^{-1}=\mathcal{I}+\mathcal{K},$$

where $\mathcal{K}:\ell_K^2\to\ell_K^2$ is a compact operator for $K=D-1,D,\ldots$.

The well-conditionedness implied by Lemma 4.1 follows from Lemma 4.4 of Olver & Townsend (2013) and is confirmed in Section 6.1 by extensive numerical experiments.

There are two remarks to be made regarding the proposed preconditioner. First, $\mathcal{W}^k$ can be interpreted as a *coarse-grid* preconditioner or *low-order discretization* preconditioner that captures the low-frequency components of the problem, leaving the high frequencies to be treated by the Krylov subspace iteration. What is also noteworthy is that it works in the frequency/coefficient space directly—there is no need to do the interpolation and transfer back and forth between the physical/value and the frequency/coefficient spaces.

Second, the diagonal preconditioner $\mathcal{R}$ given in Olver & Townsend (2013, §4.1) continues to work, despite the loss of bandedness in (2.11). On the one hand, it is apparent that the diagonal preconditioner costs less to apply. On the other hand, experiments show that the proposed preconditioner has a better chance to make the eigenvalues of J_n^k cluster.[3] Therefore, it is difficult to say which preconditioner is more effective. Whether the proposed preconditioner (significantly) outperforms the diagonal one also depends on other factors, including but not limited to how frequently GMRES is restarted, the forcing term ω^k, etc. Extensive numerical tests show that the INGU method is faster with the new preconditioner and very much so when the number of iterations allowed before GMRES restarts is not very large.

5. Further acceleration

The fast application of the Jacobians and the preconditioner are decisive in making our INGU method fast. In this section, we discuss other opportunities that may potentially allow the computation to be further accelerated.

5.1 *Mixed precision*

After a rapid development in the last couple of decades, mixed precision algorithms have earned a proven track record in accelerating iterative methods and made inroads into the tool set of our day-to-day computation (Abdelfattah *et al.*, 2021; Higham & Mary, 2022).

Tisseur analyzes the limiting accuracy and limiting residual of Newton's method in floating point arithmetic in a multiple-precision setting (Tisseur, 2001). A recent work by Kelley (2022) investigates the use of reduced precision arithmetic to solve the linearized equation for the Newton update by a direct linear solver. These works are reviewed and summarized in Higham & Mary (2022, §5). See also Algorithm 5.1 therein. The main idea is to compute the residual in a relatively high precision p_h and the Newton update in a relatively low precision p_l, while maintain the approximate solution u^k in a working precision p_w with $p_h \leq p_w \leq p_l$. In Kelley (2022), $p_h = p_w$ are chosen to be double precision and p_l single or half precisions.

Our mixed-precision implementation for the INGU method follows the same strategy. To be specific, we compute and store the residual and the solution in double precision, whereas the GMRES solve is done in single precision. Similar to what is reported in Kelley (2022), our numerical experiments show that the result from a reduced-precision implementation of GMRES does not distinguish from those done by a fixed-precision computation with double precision throughout. We observe an average gain of 20% to 30% in speed.

[3] Of course, eigenvalues clustering needs not indicate fast convergence (Greenbaum & Strakoš, 1994; Greenbaum *et al.*, 1996).

TABLE 1 *A collection of 1D nonlinear ODE boundary value problems*

equation and BCs	linearization	note
Blasius equation $u''' + uu''/2 = 0$ $u(0) = 0, u'(0) = 0, u'(L) - 1 = 0$	$\delta''' + (u\delta'' + u''\delta)/2 = 0$ $\delta(0) = 0, \delta'(0) = 0, \delta'(L) = 0$	boundary layer
Falkner–Skan equation $u''' + uu''/2 + 2\left(1 - (u')^2\right)/3 = 0$ $u(0) = 0, u'(0) = 0, u'(L) - 1 = 0$	$\delta''' + (u\delta'' + u''\delta)/2 - 4u'\delta'/3 = 0$ $\delta(0) = 0, \delta'(0) = 0, \delta'(L) = 0$	an extension of the Blasius equation
Fisher–KPP equation $u'' + u(1 - u) = 0$ $u(-4) - 1 = 0, u(4) = 0$	$\delta'' + \delta - 2u\delta = 0$ $\delta(-4) = 0, \delta(4) = 0$	a perturbed reaction-diffusion equation
fourth-order equation $u^{(4)} - u'u'' + uu''' = 0$ $u(0) = 0, u'(0) = 0,$ $u(1) - 1 = 0, u'(1) + 5 = 0$	$\delta^{(4)} - u'\delta'' - u''\delta' + u\delta''' + u'''\delta = 0$ $\delta(0) = 0, \delta'(0) = 0,$ $\delta(1) = 0, \delta'(1) = 0$	the equation of highest-order in this collection
Bratu equation $u'' + \beta e^u = 0$ $u(-1) = 0, u(1) = 0$	$\delta'' + \beta e^u \delta = 0$ $\delta(-1) = 0, \delta(1) = 0$	no solution when $\beta > 0.878$ & closed-form solution exists
Lane–Emden equation $xu'' + 2u' + xu^5 = 0$ $u(0) - 1 = 0, u'(0) = 0$	$x\delta'' + 2\delta' + 5xu^4\delta = 0$ $\delta(0) = 0, \delta'(0) = 0$	an IVP solved as a BVP and closed-form solution exists
gulf stream $u''' - \beta\left(\left(u'\right)^2 - uu''\right) - u + 1 = 0$ $u(0) - 1 = 0, u''(0) = 0,$ $u(L) - 1 = 0$	$\delta''' - \beta\left(2u'\delta' - u\delta'' - u''\delta\right) - \delta = 0$ $\delta(0) = 0, \delta''(0) = 0, \delta(L) = 0$	a conservation law holds for u and second-order boundary condition
interior layer $\epsilon u'' + uu' + u = 0$ $u(0) + 7/6 = 0, u(1) - 3/2 = 0$	$\epsilon\delta'' + u\delta' + u'\delta + \delta = 0$ $\delta(0) = 0, \delta(1) = 0$	singularly perturbed by the leading coefficient
boundary layer $\epsilon u'' + uu' - xu = 0$ $u(0) + 7/6 = 0, u'(1) - 3/2 = 0$	$\epsilon\delta'' + u\delta' + u'\delta - x\delta = 0$ $\delta(0) = 0, \delta'(1) = 0$	as above
sawtooth $\epsilon u'' + (u')^2 - 1 = 0$ $u(-1) - 0.8 = 0, u(1) - 1.2 = 0$	$\epsilon\delta'' + 2u'\delta' = 0$ $\delta(-1) = 0, \delta(1) = 0$	as above
Allen–Cahn equation $\epsilon u'' + u - u^3 - \sin(x) = 0$ $u(0) - 1 = 0, u(10) + 1 = 0$	$\epsilon\delta'' + \delta - 3u^2\delta = 0$ $\delta(0) = 0, \delta(10) = 0$	singularly perturbed steady state equation
pendulum $u'' + \sin u = 0$ $u(0) - 2 = 0, u(10) - 2 = 0$	$\delta'' + \cos u\delta = 0$ $\delta(0) = 0, \delta(10) = 0$	multiple solutions

(Continued)

TABLE 1 *Continued*

equation and BCs	linearization	note
Carrier equation $\epsilon u'' + 2(1-x^2)u + u^2 - 1 = 0$ $u(-1) = 0, u(1) = 0$	$\epsilon\delta'' + 2(1-x^2)\delta + 2u\delta = 0$ $\delta(-1) = 0, \delta(1) = 0$	as above
Painlevé equation $u'' - u^2 + x = 0$ $u(0) = 0, u(L) - \sqrt{L} = 0$	$\delta'' - 2u\delta = 0$ $\delta(0) = 0, \delta(L) = 0$	as above
Birkisson I $u'' - (\cos x)u' + u\log u = 0$ $u(0) - 1 = 0, u(\pi/2) - e = 0$	$\delta'' - (\cos x)\delta' + (\log u + 1)\delta = 0$ $\delta(0) = 0, \delta(\pi/2) = 0$	closed-form solution exists
Birkisson II $u'' - u' + e^{2x}u + u^2 = \sin^2(e^x)$ $u(0) - \sin 1 = 0,$ $u(5/2) - \sin\left(e^{5/2}\right) = 0$	$\delta'' - \delta' + e^{2x}\delta + 2u\delta = 0$ $\delta(0) = 0, \delta(5/2) = 0$	as above
Birkisson III $u'' + 18(u - u^3) = 0,\ x \in [-1, 1]$ $u(-1) + \tanh 3 = 0,$ $u(0) - \tanh 0 = 0$	$\delta'' + 18(\delta - 3u^2\delta) = 0$ $\delta(-1) = 0, \delta(0) = 0$	interior point condition

There are a few more opportunities for further GMRES speed boost by mixed-precision arithmetic. First, it has been shown in Simoncini & Szyld (2003); Van Den Eshof & Sleijpen (2004); Giraud *et al.* (2007) that matrix-vector products can be done in increasingly reduced precision without degrading the overall accuracy of the entire computation. In addition, the orthonormalization step in GMRES can also be performed in a reduced precision (Gratton *et al.*, 2019). Hence, we could implement the matrix-vector multiplication and the orthonormalization in, for example, half precision using fp16 or bfloat16. Second, we can replace a single GMRES solve by an iterative refinement solve by performing reduced-precision GMRES as an inner solver for the corrections (Turner & Walker, 1992). Third, we could have started our Newton's method with a low precision, e.g., half precision, and only upgrade precision once Newton's method converges to the current precision. We, however, choose not to pursue these enhancements in this work for a few reasons. First, using a reduced precision for the entire GMRES solve is the easiest to implement, but gain the most. Also, our primary goal is to demonstrate that Newton–GMRES method can be done in mixed precision, which does not seem previously to have appeared in the literature. The possible enhancements listed above are out of the scope of the current investigation. Second, though the limiting accuracy and residual would stay the same, the effect of reduced precision on the convergence rate is not fully understood for some of these enhancements. The quadratic convergence may be at stake. Finally, hardware support for half precision is not as widely available in CPUs as in GPUs at the time of writing. Moreover, JULIA currently supports half precision only through software emulation at, inevitably, a huge cost of speed. We save this line of research for the future.

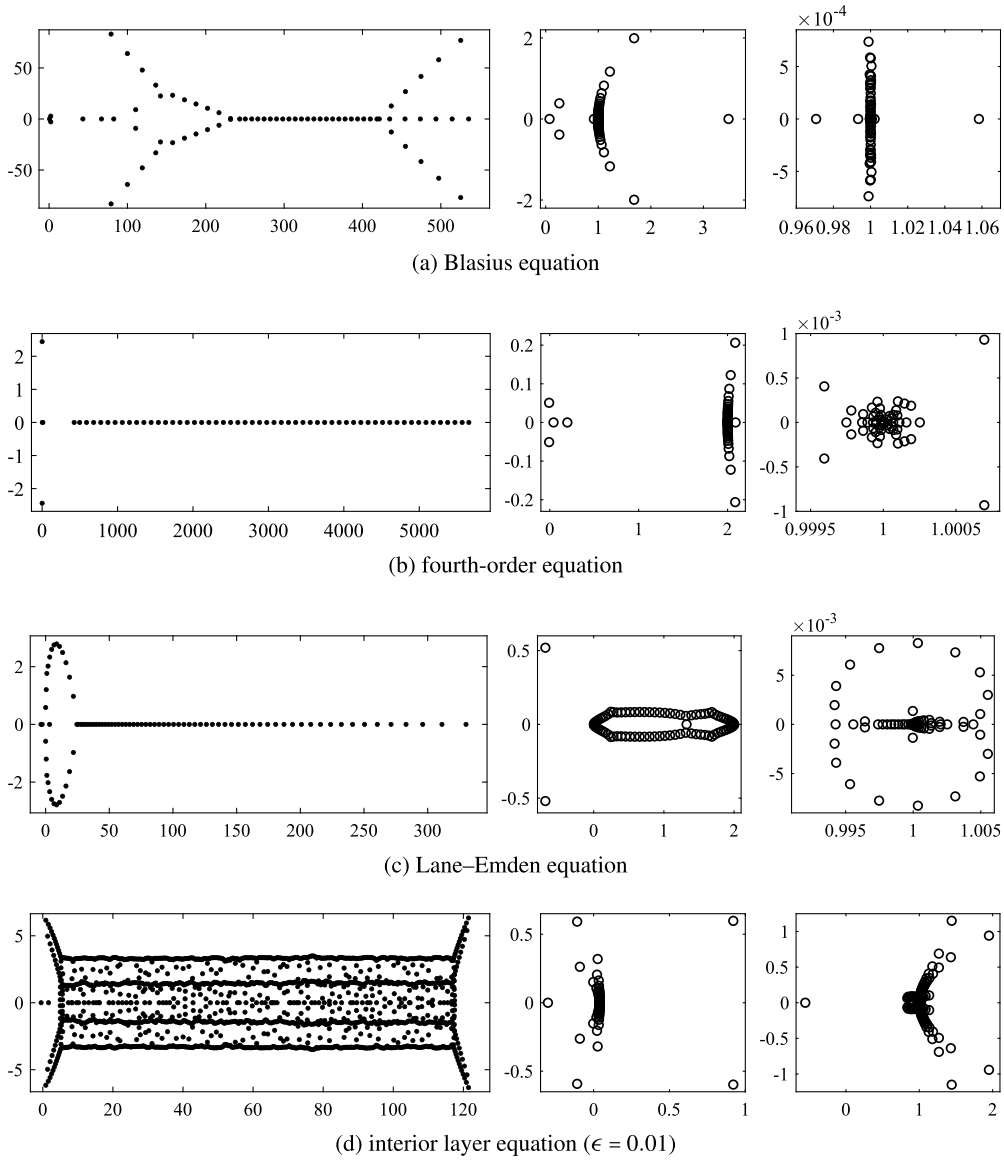
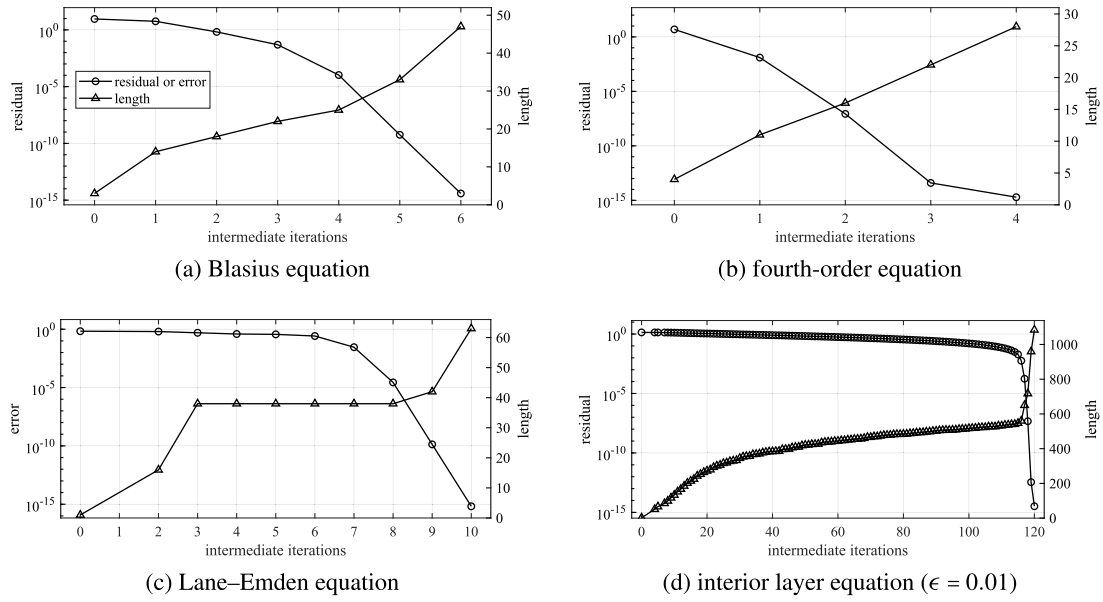


FIG. 1. The eigenvalue distributions of the original systems (left panes), those preconditioned by the diagonal preconditioner (middle panes) and by the almost-banded preconditioners (right panes) for the last intermediate iteration before the termination of the Newton process. Note the conspicuous difference in the axis scales across each triptych.

5.2 *Krylov subspace acceleration*

GMRES also stands a chance for further acceleration with various Krylov subspace techniques. In Parks *et al.* (2006), the authors suggest that the cost of constructing the Krylov subspace in an iteration of the Newton–GMRES method can be reduced by salvaging the Krylov subspaces of the previous iterations.



(a) Blasius equation

(b) fourth-order equation

(c) Lane–Emden equation

(d) interior layer equation ($\epsilon = 0.01$)

FIG. 2. Convergence (vertical axis on the left) and the increase of the solution length (vertical axis on the right) for the four selected problems.

However, our experiments show that the gain acquired from this Krylov subspace recycling strategy is very marginal in the current context, if at all, as the Krylov subspaces vary very quickly across Newton iterations.

Recently, fast randomized sketching is utilized to speed up the subspace projection in GMRES (Nakatsukasa & Tropp, 2021). However, we found that the sketched GMRES (sGMRES) can hardly accelerate the INGU method. As pointed out in Nakatsukasa & Tropp (2021, §8.2), sGMRES can hardly give a speed boost if the matrix-vector multiplication outweighs other parts of the algorithm, which is exactly the case in the INGU method—though it is done by FFTs, the Jacobian-vector multiplication is still the most costly part. Therefore, sGMRES has a very large break-even in our experiments and no gain in speed is seen if $n \lesssim 10^4$. Unless a problem is singularly perturbed (see Section 6.4), there is usually no need for such large degrees of freedom in a 1D problem.

Meanwhile, sGMRES is unable to handle ill-conditioned systems even when GMRES succeeds (Nakatsukasa & Tropp, 2021, §8.2). This also rules out the possibility of applying it to singularly perturbed problems, since the condition number of such a problem is usually extremely large.

Hence, these Krylov subspace acceleration techniques are not included in the experiments shown in Section 6. However, we believe that these techniques have much bigger potentials for problems in higher spatial dimensions, particularly for those not singularly perturbed.

6. Numerical experiments

We collect 17 univariate nonlinear ODE boundary value problems from various sources, such as Birkisson (2013); Driscoll *et al.* (2014), and use them as a test bank for the INGU framework. These problems and their linearizations are gathered in Table 1 with a one-liner note for each. Five of these

problems have a closed-form solution—the solutions to the Bratu and Lane–Emden equations can be found in standard texts of applied mathematics, while those to the Birkisson equations are given in Birkisson (2013). The residual and error are measured in the 2-norm, and we choose to have GMRES restarted after $r = n/100$ iterations. In case of $r < 20$ or $r > 150$, we simply take $r = 20$ and $r = 150$, respectively. All the numerical experiments are performed in JULIA v1.8.3 on a laptop with a 4 core 2.8 Ghz Intel Core i7 CPU and 16GB RAM. The execution times are obtained using `BenchmarkTools.jl`.

6.1 *Preconditioning*

To demonstrate the effectiveness of the preconditioner, we solve the Blasius, the fourth-order, the Lane–Emden and the interior layer equations and compare the eigenvalue distributions of J_n^k with those of J_n^k preconditioned with the diagonal preconditioner and those of $J_n^k \left(W_n^k\right)^{-1}$ in Fig. 1 for the systems arise in the last intermediate iteration of each problem. These four problems are typical and representative of the entire collection. The Blasius equation features a mild boundary layer formed by a physically meaningful no-slip boundary condition rather than a boundary layer caused by singular perturbation. The fourth-order equation is the one of highest order in this collection. The Lane–Emden equation is an ODE initial value problem that we solve by regarding it as an ODE boundary value problem. A closed-form solution of the Lane–Emden equation is known, which helps in measuring the solution error instead of the residual. The equation labeled as interior layer has an interior layer caused by singular perturbation in the leading order term with $\epsilon = 0.01$ and this interior layer results in a solution with length greater than 1000 for a complete resolution.

The marked difference in the axis scales between the left panes (the original systems) and the middle and right panes (the systems preconditioned by the diagonal and almost-banded preconditioners, respectively) underscores the significant improvement in conditioning. For the first three examples, the eigenvalues resulting from the almost-banded preconditioner are much more closely clustered compared to those from the diagonal preconditioner. In the interior layer example, eigenvalues produced by the almost-banded preconditioner predominantly cluster around unity in the complex plane. In contrast, the eigenvalues obtained with the diagonal preconditioner center around the origin, which suggests that GMRES may converge at a very slow rate. This is because when $Ax = b$ is solved by GMRES the relative residual $\left\|r_n\right\| / \|b\|$ after n iterations is bounded by $\inf_{p \in P_n} \|p(A)\|$, where $P_n = \{\text{polynomials } p \text{ of degree } \leq n \text{ with } p(0) = 1\}$. With A's eigenvalues all being close to zero, it is difficult to diminish $\|p(A)\|$ (Trefethen & Bau, 1997, §35). The superiority of the almost-banded preconditioner observed in these examples is consistent throughout the rest of the collection.

6.2 *Convergence*

We take the four problems above again for examples to show the typical convergence of the INGU method. In Fig. 2, we show the convergence history by plotting the residual $\mathcal{G}(u^k)$ or absolute error and the lengths of the approximate solutions for the TR–contravariant method. The other two global methods produce very similar results. The residual/error and the solution length can be read off from the y-axes on the left and the right, respectively. The x-axis indicates the number of intermediate iterations. A marker signals the start of an outer iteration.

The residual curves for all four problems show that the convergence usually evolves with two phases—the first phase is characterized by the relatively level trajectory that corresponds to the slow convergence in the global Newton stage and the second phase features a steep descent of the residual/error

that is the consequence of the fast convergence of the local Newton stage. The pattern is most obvious for the interior layer problem with the first 115 intermediate iterations for the global Newton search until the quadratic convergence kicks in at about the 5th to last iteration.

The evolution of the solution lengths matches the decay of the residual in that the lengths increase only moderately until the local Newton phase is reached where the lengths grow very quickly for much improved resolution.

6.3 *Speed and accuracy*

Now we examine the INGU method for its speed and accuracy by benchmarking against plain implementations of the three global methods. In these plain implementations with only double precision throughout, the outer and intermediate loops are retained, whereas GMRES is replaced by LU factorization, resulting in the exact Newton condition being always satisfied. We refer to this plain approach as the exact Newton LU ultraspherical method (ENLU). The ENLU framework is adopted, for example, by CHEBFUN (Driscoll *et al.*, 2014) and APPROXFUN[4] (Olver, 2019).

In an ENLU implementation, solving (2.11) by LU or QR costs $\mathcal{O}(2n^3/3)$ and $\mathcal{O}(4n^3/3)$ flops, respectively. For INGU, the dominant cost for solving (2.11) is the fast Jacobian-vector multiplication that entails $\mathcal{O}(\kappa n \log n)$ flops, where we assume that GMRES finds an inexact solution in κ iterations. Note that this outweighs the cost of applying the preconditioner W_n^k—one QR factorization in $\mathcal{O}(n \log n)$ flops and back substitution in $\mathcal{O}(\kappa n\sqrt{\log n})$ flops. Hence, INGU is expected to have advantage for large scale problems.

The contrast may be even more stark for the construction of (2.11). If for $\lambda \geq 2$ any of $a^\lambda(x)$ is variable, either (2.3) or (2.4) must be employed for the explicit construction of $\mathcal{M}_\lambda[a^\lambda]$. In case of $a^\lambda(x)$ depending on $u(x)$, this cost is $\mathcal{O}(20n^3/3)$ and $\mathcal{O}(37n^3/3)$, respectively. In our collection, Blasius, Falkner–Skan, fourth-order and gulf stream equations fall into this category. See the middle column in Table 1 for the linearized equations. The significance of this cost can be perceived by realizing that it would be the cost of solving (2.11) by LU 10 or 18 times. Even in the absence of such high-order terms, the construction of (2.11) still costs $\mathcal{O}(n^2)$ flops. Quite the contrary, INGU never constructs $\mathcal{M}_\lambda[a^\lambda]$ or any other operators explicitly—all the calculation is done on the fly. The only matrix that INGU explicitly forms is the preconditioner W_n^k at a relatively small cost of $\mathcal{O}(n\sqrt{\log n})$ flops. Thus, clear advantage is expected from INGU in matrix construction, especially when there is a solution-dependent variable coefficient in the higher-order terms of the linearized problem.

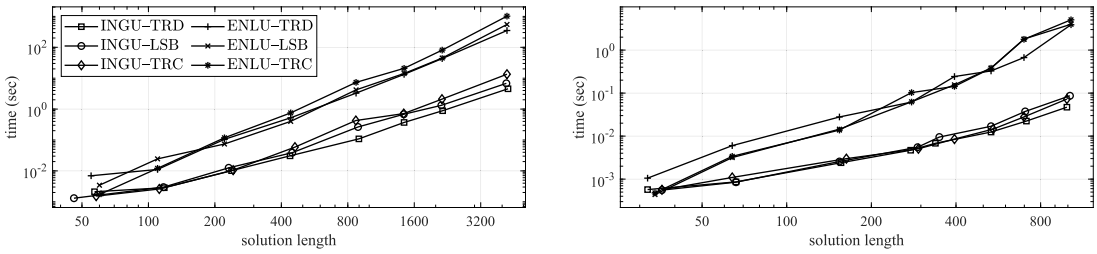
In terms of storage, INGU is also advantageous compared to ENLU, i.e., $\mathcal{O}(n\sqrt{\log n})$ versus $\mathcal{O}(n^2)$.

We solve the entire collection using both INGU and ENLU implementations. For the ENLU implementations, LU is used for linear solve and $\mathcal{M}_\lambda[a^\lambda]$, if any, is constructed via (2.3). The results are listed in Table 2, where only the data for TR–contravariant (TRC) are displayed as the other two global methods give similar results. The first column contains the values of the parameters used, if any, right beneath the equation name. Columns 2 and 4 list the accuracy that INGU–TRC and ENLU–TRC can achieve and the corresponding execution time. The accuracy is measured by the absolute residual $\mathcal{G}(u^k)$ for the approximate solution u^k or the absolute error at the termination of the outer iteration in case of a closed-form solution being available. Column 5 shows the speedup that is the ratio of the ENLU–TRC execution time to that of INGU–TRC. We also repeat the experiments in double precision throughout to exclude the benefits gained from the use of mixed-precision arithmetic. See Column 3 for the accuracy and time and Column 6 for the speedups. The last column gives the final solution length.

[4] In APPROXFUN, QR, instead of LU, is used, causing twice the cost for linear solve.

TABLE 2 *INGU vs ENLU in TRC implementation*

equation	INGU–TRC	INGU–TRC (double)	ENLU–TRC	speedup	speedup (double)	length
Blasius ($L = 10$)	3.88e−15 7.11e−4	1.00e−14 1.90e−3	9.93e−15 1.16e−1	163.11	61.20	47
Falkner–Skan ($L = 10$)	3.77e−15 8.94e−4	2.84e−14 1.34e−3	1.43e−14 1.63e−2	18.24	12.17	28
Fisher–KPP	2.88e−15 1.17e−3	2.31e−15 1.25e−3	4.39e−15 3.46e−3	2.96	2.77	62
fourth-order	1.93e−15 9.35e−4	2.76e−15 1.14e−3	3.58e−15 7.77e−3	8.31	6.84	28
Bratu ($\beta = 0.875$)	3.33e−15 1.11e−3	6.00e−15 1.18e−3	7.11e−15 1.45e−3	1.31	1.23	39
Lane–Emden	6.66e−16 3.79e−3	3.89e−16 5.06e−3	4.44e−16 3.36e−2	8.87	6.64	63
gulf stream ($\beta = -0.1, L = 35$)	6.24e−13 3.03e−3	2.51e−12 3.11e−3	4.70e−12 9.00e−2	29.70	29.00	62
interior layer ($\epsilon = 0.01$)	3.28e−15 3.63e−1	1.67e−15 9.33e−1	6.50e−15 6.53	17.97	7.00	1084
boundary layer ($\epsilon = 0.01$)	2.04e−15 7.20e−3	4.24e−15 1.34e−2	2.68e−15 1.25e−1	17.38	9.33	248
sawtooth ($\epsilon = 0.05$)	1.67e−15 1.01e−2	7.87e−16 2.35e−2	6.85e−16 1.71e−1	16.96	7.27	425
Allen–Cahn ($\epsilon = 2$)	8.51e−16 2.81e−3	1.45e−16 4.71e−3	1.45e−16 1.13e−2	4.02	2.40	79
pendulum	1.22e−14 8.43e−4	1.94e−15 1.19e−3	3.71e−15 1.95e−3	2.31	1.64	43
Carrier ($\epsilon = 0.01$)	8.35e−16 9.79e−3	4.96e−16 1.79e−2	2.45e−16 3.59e−2	3.67	2.01	225
Painlevé ($L = 10$)	3.27e−14 7.57e−4	1.30e−14 1.12e−3	3.66e−14 2.63e−3	3.48	2.35	46
Birkisson I	8.88e−16 9.52e−4	1.33e−15 1.18e−3	1.33e−15 1.75e−3	1.84	1.49	25
Birkisson II	4.39e−15 2.22e−3	4.55e−15 3.00e−3	3.77e−15 5.49e−3	2.48	1.83	52
Birkisson III	2.55e−15 2.74e−3	7.77e−16 3.20e−3	3.66e−15 5.66e−3	2.07	1.77	88



(a) interior layer equation (b) sawtooth equation

FIG. 3. Solution length versus execution time for singularly perturbed problems with various ϵ.

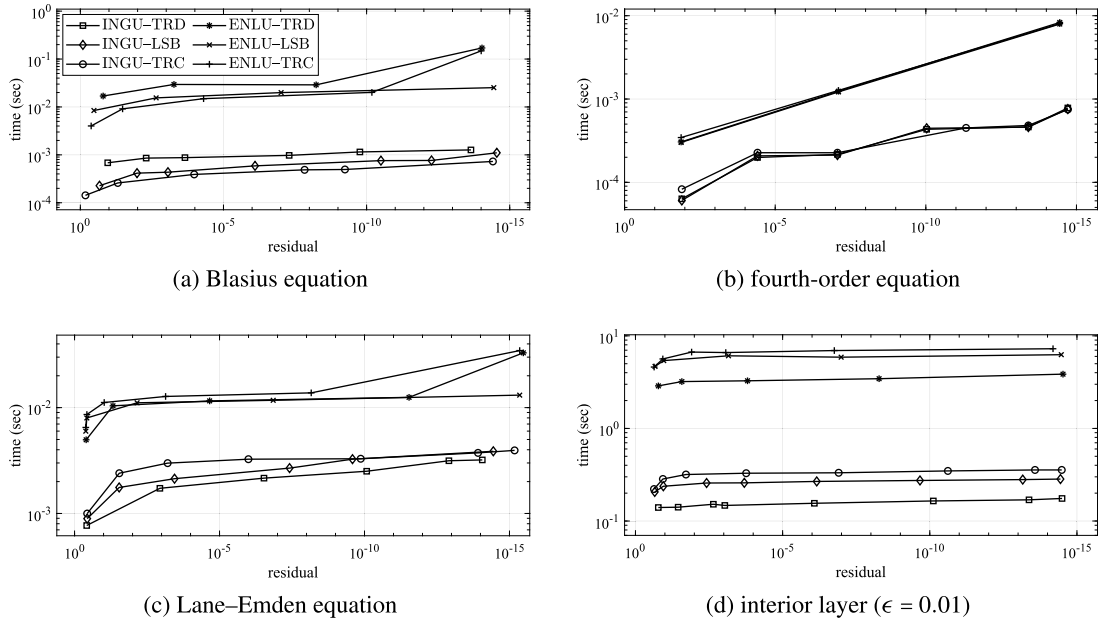
We immediately identify the large speedups in Blasius, Falkner–Skan, fourth-order and gulf stream equations, which are attributed to the considerable cost for constructing $\mathcal{M}_\lambda[a^\lambda]$ ($\lambda \geq 2$) in ENLU, even though these problems are not large at all. As expected, large speedups also occur in the problems with long solutions, such as the interior layer, boundary layer, sawtooth equations, where INGU solves the linear systems for inexact Newton steps much faster than ENLU does for exact ones. For the rest of the collection, which are small scale problems with no need to construct high order multiplication operators, the speedups range from marginal to modest. These speedups are mainly due to the relatively cheaper matrix construction of $\mathcal{O}(n\sqrt{\log n})$ flops in INGU versus $\mathcal{O}(n^2)$ flops in ENLU.

To get a sense of how the degrees of freedom n plays its role in the speedup, we vary ϵ in the interior layer and sawtooth equations and record the solution lengths and the execution time for each ϵ until the problem is too singularly perturbed that the simple initial iterate for the outer loop described in Section 2.5 leads to divergence or different solutions. Figure 3 shows the result. It can be seen that INGU has a clear speed advantage and this advantage is increasingly conspicuous as n grows. For the largest n shown here, the speedups are roughly 77 and 66, respectively. Note that in these two examples there is no expensive matrix construction involved, and the speedup is solely brought about by the fast solution of INGU.

Finally, we show in Fig. 4 the accuracy-versus-time curves of the four problems discussed in Sections 6.1 and 6.2 for all three pairs of INGU and ENLU implementations. Multiple bailout tolerances are used to obtain roughly equispaced residuals or errors in logarithm along with the corresponding execution times. A couple of observations can be made. First, same as TRC (see Table 2) all other global implementations also reach an accuracy close to the machine precision. Second, for a same accuracy goal, the three INGU implementations have comparable execution times, and their speedups over the ENLU counterparts are roughly uniform across almost all target accuracies only except in the fourth-order problem. For the fourth-order problem, the speedup is marginal when the tolerances are large, and it ramps up as the tolerance is reduced.

6.4 *Singularly perturbed problems*

So far, we have not yet chosen particularly small ϵ for the singularly perturbed equations. As ϵ becomes smaller, difficulty arises as the region(s) outside which the initial iterate would fail to converge shrinks rapidly. To produce a quality initial iterate, we resort to the approach of pseudo-arclength continuation (Nocedal & Wright, 2006; Birkisson, 2013; Kelley, 2018). Specifically, we solve the same equation, but with a much larger ϵ for which a simple initial iterate usually suffices, e.g., the polynomial of the lowest degree that satisfies the boundary conditions. This easy problem can be solved by the INGU method up to a low accuracy, and the solution, along with the corresponding ϵ, serves as the starting point for the



(a) Blasius equation

(b) fourth-order equation

(c) Lane–Emden equation

(d) interior layer ($\epsilon = 0.01$)

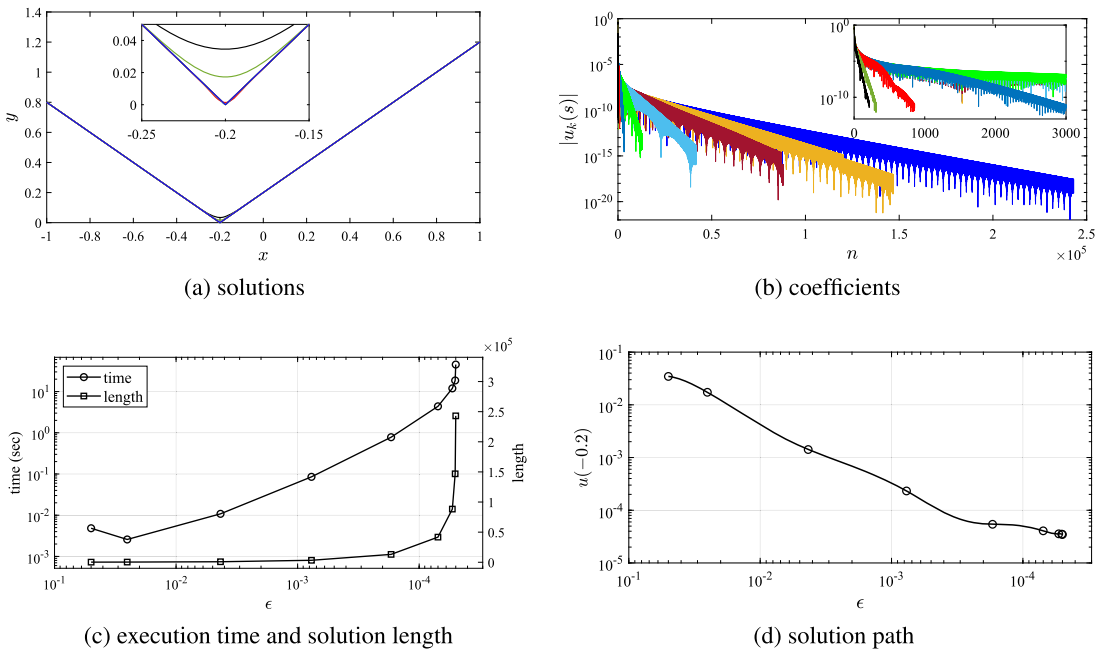
FIG. 4. Residual or error versus execution time for the selected problems.

continuation process. It is then followed by parameterizing the solution $u(x, s) = \sum_{j=0}^{n(s)-1} u_j(s)T_j(x)$ and the perturbation parameter $\epsilon(s)$ by the arclength s along the so-called solution path and tracing this path by the common predictor-corrector method (Allgower & Georg, 2003; Nocedal & Wright, 2006) in order to get close to the target ϵ. In each predictor-corrector iteration, we leave the solution path by marching along the predictor direction before successive corrector iterations bring us (almost) back onto the path so that we have the solution to the original equation, but with an ever smaller ϵ. The path tracing ceases one predictor-corrector iteration before the target ϵ is overshot. Note that (1) these intermediate solutions usually need not to be calculated to high accuracy, as long as they finally lead to a good initial iterate; (2) both the predictor and the corrector steps are obtained by solving the corresponding expanded equations, and since these expanded equations are formed by augmenting (2.11) with one more row and column, the solution can be accelerated by the fast multiplication, the preconditioner and the mixed-precision arithmetic as above.

We take for example the sawtooth equation with $\epsilon = 5 \times 10^{-5}$ to demonstrate how an average singularly perturbed equation is solved. We start off by solving the sawtooth equation with $\epsilon = 5 \times 10^{-2}$ for which the linear polynomial that satisfies the boundary conditions is good enough as the initial iterate. Setting $s = 0$ for this solution and marching with the predictor-corrector method as described above, we obtain a sequence of 7 more intermediate solutions on the path and the corresponding ϵ, before reaching $\epsilon = 5 \times 10^{-5}$. For each predictor step, the corrector iteration stops once the last update in the current corrector is smaller than 10^{-2} in a relatively sense. See columns 2–8 in Table 3 for ϵ along with the length of the corresponding intermediate solution and the execution time for its calculation. These data are also plotted in Fig. 5c. Finally, the last intermediate solution is fed into the INGU method as the initial iterate. The length of the solution to the original problem and the execution time are appended in the last column of Table 3, and the residual of this final solution is about 4.07×10^{-15}. Note that it takes about 45 seconds

TABLE 3 *The intermediate and final solutions to the sawtooth equation on the solution path for progressively smaller* ϵ

ϵ	5.00e−2	2.52e−2	4.31e−3	7.69e−4	1.70e−4	7.00e−5	5.33e−5	5.06e−5	5.00e−5
length	214	311	857	3,273	13,071	41,733	88,296	146,868	243,065
time (sec)	4.83e−3	2.58e−3	1.08e−2	8.53e−2	7.81e−1	4.38	1.20e1	1.86e1	4.52e1



(a) solutions (b) coefficients

(c) execution time and solution length (d) solution path

FIG. 5. Solving sawtooth equation with $\epsilon = 5 \times 10^{-5}$.

to calculate the final solution whose length is 243, 065! Even if we take into account the pre-computation of the intermediate solutions, the total execution time is still as little as 82 seconds, which gives a good sense of the speed that the INGU method offers. The scale of the computation in this example almost exhausts the RAM of the machine on which this experiment is carried out. If the memory were large enough, we should be able to solve the sawtooth equation that is even more singularly perturbed. As expected, ENLU fails to produce the fifth intermediate solution due to a quick drain on the memory.

The sequence of the solutions, including the intermediate ones and the one to the original equation, are shown in Fig. 5a, with a close-up displaying the solutions at the turning point $x = -0.2$. As expected, the solutions become more pointed as ϵ diminishes. The coefficients of these solutions are shown in Fig. 5b with a close-up for the first few intermediate solutions that are too short to be seen in the master plot. We also include a plot of the solution path in Fig. 5d to show the evolution of the value of the solution at $x = -0.2$ versus that of ϵ.

7. Concluding remarks

The success of the proposed INGU framework in solving nonlinear equations demonstrates that the ultraspherical spectral method is still a powerful tool beyond the linear regime, thanks to the structured operators of differentiation, multiplication and conversion. The source code of our implementation is publicly available at Qin (2023).

Can we solve nonlinear ODEs in the same vein by the collocation-based pseudospectral method? The answer is probably no. The value-based approximation of differential operators can also be quickly applied, since fast differentiation can also be effected via FFT and variable coefficients are represented as diagonal matrices. However, the construction of an efficient preconditioner to expedite the convergence of Krylov subspace methods remains unknown. This highlights an advantage of the ultraspherical spectral method over the pseudospectral approach.

Immediate applications of INGU include but not limited to nonlinear ODEs arising from solving nonlinear time-dependent PDEs (Cheng & Xu, 2023) and nonlinear eigenvalue problems of operators with eigenvector nonlinearity (NEPv) (Upadhyaya, 2021, §2.3.3). The INGU framework can also be extended from ultraspherical or coefficient-based spectral methods for integral equations (Slevinsky & Olver, 2017), convolution integral equations (Xu & Loureiro, 2018; Hale, 2019) and fractional integral equations (Hale & Olver, 2018) to their respective nonlinear counterparts, as the infinite operators in these equations are also structured.

There are both challenges and opportunities towards extending the ultraspherical spectral method to solving nonlinear differential equations in higher spatial dimensions as generalized Sylvester equations (Townsend & Olver, 2015) and tensor decomposition are involved (Strössner & Kressner, 2023).

Acknowledgements

We would like to thank Lu Cheng and Sheehan Olver for their valuable feedback on an early draft of this paper, which led us to improve our work. We thank Keaton Burns, Marco Fasondini, Ioannis Papadopoulos, Geoff Vasil and Marcus Webb for comments and Joel Tropp for the sGMRES code. We are grateful to two very helpful anonymous referees and the associate editor Nick Trefethen for the extremely constructive advice and suggestions.

References

Abdelfattah, A., Anzt, H., Boman, E. G., Carson, E., Cojean, T., Dongarra, J., Fox, A., Gates, M., Higham, N. J., Li, X. S., et al. (2021) A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *Int. J. High Perform. Comput. Appl.*, **35**, 344–369.

Allgower, E. L. & Georg, K. (2003) *Introduction to Numerical Continuation Methods*. Philadelphia: SIAM.

Atkinson, K. & Han, W. (2005) *Theoretical Numerical Analysis*, vol. 39. Berlin: Springer.

Aurentz, J. L. & Trefethen, L. N. (2017) Chopping a Chebyshev series. *ACM Trans. Math. Softw.*, **43**, 1–21.

Birkisson, A. (2013) Numerical solution of nonlinear boundary value problems for ordinary differential equations in the continuous framework. *Ph.D. Thesis*. Oxford University, UK.

Birkisson, A. & Driscoll, T. A. (2012) Automatic Fréchet differentiation for the numerical solution of boundary-value problems. *ACM Trans. Math. Softw.*, **38**, 1–29.

Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D. & Brown, B. P. (2020) Dedalus: a flexible framework for numerical simulations with spectral methods. *Phys. Rev. Res.*, **2**, 023068.

Cheng, L. & Xu, K. (2023) Solving time-dependent PDEs with the ultraspherical spectral method. *J. Sci. Comput.*, **96**, 70.

Cheng, L. & Xu, K. (2024) Understanding the ultraspherical spectral method. In Preparation.
Deuflhard, P. (2005) *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*, vol. 35. Berlin: Springer Science & Business Media.
Driscoll, T. A., Hale, N. & Trefethen, L. N. (2014) *Chebfun Guide*. Oxford: Pafnuty Publications.
Eisenstat, S. C. & Walker, H. F. (1994) Globally convergent inexact Newton methods. *SIAM J. Optim.*, **4**, 393–422.
Eisenstat, S. C. & Walker, H. F. (1996) Choosing the forcing terms in an inexact Newton method. *SIAM J. Sci. Comput.*, **17**, 16–32.
Frigo, M. & Johnson, S. G. (2005) The design and implementation of FFTW3. *Proc. IEEE*, **93**, 216–231. Special issue on 'Program Generation, Optimization, and Platform Adaptation'.
Giraud, L., Gratton, S. & Langou, J. (2007) Convergence in backward error of relaxed GMRES. *SIAM J. Sci. Comput.*, **29**, 710–728.
Golub, G. H. & Van Loan, C. F. (2013) *Matrix Computations*, 4th edn. Baltimore MD: JHU Press.
Gratton, S., Simon, E., Titley-Peloquin, D. & Toint, P. (2019) Exploiting variable precision in GMRES. arXiv preprint arXiv:1907.10550.
Greenbaum, A., Pták, V. & Strakoš, Z. (1996) Any nonincreasing convergence curve is possible for GMRES. *SIAM J. Matrix Anal. Appl.*, **17**, 465–469.
Greenbaum, A. & Strakoš, Z. (1994) Matrices that generate the same Krylov residual spaces. *Recent Advances in Iterative Methods*. New York: Springer, pp. 95–118.
Griewank, A. & Walther, A. (2008) *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia: SIAM.
Hale, N. (2019) An ultraspherical spectral method for linear Fredholm and Volterra integro-differential equations of convolution type. *IMA J. Numer. Anal.*, **39**, 1727–1746.
Hale, N. & Olver, S. (2018) A fast and spectrally convergent algorithm for rational-order fractional integral and differential equations. *SIAM J. Sci. Comput.*, **40**, A2456–A2491.
Higham, N. J. & Mary, T. (2022) Mixed precision algorithms in numerical linear algebra. *Acta Numer.*, **31**, 347–414.
Huang, Z. & Boyd, J. P. (2016) Bandwidth truncation for Chebyshev polynomial and ultraspherical/Chebyshev Galerkin discretizations of differential equations: restrictions and two improvements. *J. Comput. Appl. Math.*, **302**, 340–355.
Kelley, C. T. (1995) *Iterative Methods for Linear and Nonlinear Equations*. Philadelphia: SIAM.
Kelley, C. T. (2018) Numerical methods for nonlinear equations. *Acta Numer.*, **27**, 207–287.
Kelley, C. T. (2022) Newton's method in mixed precision. *SIAM Rev.*, **64**, 191–211.
Nakatsukasa, Y. & Tropp, J. A. (2021) Fast & accurate randomized algorithms for linear systems and eigenvalue problems. arXiv preprint arXiv:2111.00113.
Naumann, U. (2011) *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. Philadelphia: SIAM.
Nocedal, J. & Wright, S. J. (2006) *Numerical Optimization*, 2nd edn. New York: Springer.
Olver, S. (2019) *ApproxFun.jl*. https://github.com/JuliaApproximation/ApproxFun.jl. *Software Package*.
Olver, S. & Townsend, A. (2013) A fast and well-conditioned spectral method. *SIAM Rev.*, **55**, 462–489.
Parks, M. L., De Sturler, E., Mackey, G., Johnson, D. D. & Maiti, S. (2006) Recycling Krylov subspaces for sequences of linear systems. *SIAM J. Sci. Comput.*, **28**, 1651–1674.
Powell, M. J. D. (1970) A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations* (P. Rabinowitz, ed). London: Gordon and Breach, pp. 87–114.
Qin, O. (2023) *INGU*. https://github.com/ouyuanq/INGU.git. *Software Package*.
Simoncini, V. & Szyld, D. B. (2003) Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput.*, **25**, 454–477.
Slevinsky, R. M. & Olver, S. (2017) A fast and well-conditioned spectral method for singular integral equations. *J. Comput. Phys.*, **332**, 290–315.
Strössner, C. & Kressner, D. (2023) Fast global spectral methods for three-dimensional partial differential equations. *IMA J. Numer. Anal.*, **43**, 1519–1542.

Tisseur, F. (2001) Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, **22**, 1038–1057.

Townsend, A. (2014) Computing with functions in two dimensions. *Ph.D. Thesis*. UK: Oxford University.

Townsend, A. & Olver, S. (2015) The automatic solution of partial differential equations using a global spectral method. *J. Comput. Phys.*, **299**, 106–123.

Trefethen, L. N. & Bau III, D. (1997) *Numerical Linear Algebra*, vol. 50. Philadelphia: SIAM.

Turner, K. & Walker, H. F. (1992) Efficient high accuracy solutions with GMRES(m). *SIAM J. Sci. Stat. Comput.*, **13**, 815–825.

Upadhyaya, P. (2021) Numerical algorithms for nonlinear eigenproblems with eigenvector nonlinearities. *Ph.D. Thesis*. KTH Royal Institute of Technology.

Van Den Eshof, J. & Sleijpen, G. L. G. (2004) Inexact Krylov subspace methods for linear systems. *SIAM J. Matrix Anal. Appl.*, **26**, 125–153.

Xu, K. & Loureiro, A. F. (2018) Spectral approximation of convolution operators. *SIAM J. Sci. Comput.*, **40**, A2336–A2355.

Appendix A. Post-processing algorithms for the global Newton methods

We list in this appendix the algorithms of the post-processing step for each of the global Newton methods of Section 2.4. Details, including the values of the parameters, are given so that the results we report in Section 6 can be reproduced when these algorithms are plugged into line 7 of Algorithm 1. Particularly, Algorithm 3 is mainly drawn from Algorithm 11.5 and Procedure 11.6 in Nocedal & Wright (2006). Since no literature is found to have a discussion on the determination of the forcing term ω^{k+1} for TR–dogleg, we copy the strategy from the line search method. Line 24 and lines 25–26 are borrowed from Eisenstat & Walker (1996) and Kelley (1995), respectively. The main body of Algorithm 4 is drawn from Algorithm INB in Eisenstat & Walker (1996), except that line 11 is taken from Kelley (1995) and the norm in line 15 is not squared. Algorithm 5 is a combination of the global and local inexact Newton–RES methods in Deuflhard (2005, §2 & §3). The 2-norm is used throughout these algorithms.

Algorithm 3 POSTPROCESS: TR–dogleg

Inputs: The current approximate solution u^k, nonlinear operator $\mathcal{G}$, residual vector f^k, Jacobian J_n^k and its transpose $\left(J_n^k\right)^T$, inexact Newton step δ^k, current size of the trust region Δ^k, termination tolerance η.
Outputs: A new approximate solution u^{k+1}, a new size of the trust region Δ^{k+1}, a new forcing term ω^{k+1}.

$\bar{\Delta} = 100,\ \rho_a = 0.25,\ \rho_b = 0.75,\ \omega_{\max} = 0.1$ ▷ initialization
if $\|\delta^k\| \le \Delta^k$ **then**
  $\tilde{\delta}^k = \delta^k$ ▷ inexact Newton step in the trust region
else
  $g^k = \left(J_n^k\right)^T f^k$ ▷ gradient of the merit function
  $\delta^C = -\dfrac{\|g^k\|^2}{\|J_n^k g^k\|^2} g^k$ ▷ Cauchy point
  if $\|\delta^C\| \ge \Delta^k$ **then** ▷ Cauchy point out of the trust region
    $\tilde{\delta}^k = -\dfrac{\Delta^k}{\|g^k\|} g^k$ ▷ largest step along Cauchy direction
  else
    Find ν s.t. $\|\delta^C + \nu(\delta^k - \delta^C)\| = \Delta^k$. ▷ dogleg strategy
    $\tilde{\delta}^k = \delta^C + \nu(\delta^k - \delta^C)$
  end if
end if
$\rho = \dfrac{\|f^k\|^2 - \|\mathcal{G}(u^k + \tilde{\delta}^k)\|^2}{\|f^k\|^2 - \|f^k + J_n^k \tilde{\delta}^k\|^2}$ ▷ ratio of actual reduction to predicted reduction
if $\rho < \rho_a$ **then**
  $\Delta^{k+1} = \dfrac{\left\|\tilde{\delta}^k\right\|}{4}$ ▷ reduction of the trust region
else if $\rho > \rho_b$ & $\|\tilde{\delta}^k\| = \Delta^k$ **then**
  $\Delta^{k+1} = \min(\bar{\Delta},\ 2\Delta^k)$ ▷ extension of the trust region
else
  $\Delta^{k+1} = \Delta^k$ ▷ initial size of trust region $\Delta^0 = 0.1$
end if
if $\rho \ge \rho_a$ **then**
  $u^{k+1} = u^k + \tilde{\delta}^k,\ f^{k+1} = \mathcal{G}(u^{k+1})$ ▷ step accepted
  $\omega^{k+1} = 0.9\dfrac{\|f^{k+1}\|^2}{\|f^k\|^2}$ ▷ initial forcing term $\omega^0 = 0.1$
  $\omega^{k+1} = \max\left(\omega^{k+1},\ \dfrac{\eta}{2\|f^{k+1}\|}\right)$ ▷ safeguard for oversolving
  $\omega^{k+1} = \min\left(\omega^{k+1},\ \omega_{\max}\right)$ ▷ safeguard for too much inexactness
else
  $u^{k+1} = u^k,\ f^{k+1} = f^k,\ \omega^{k+1} = \omega^k$ ▷ no change
end if
return u^{k+1}, Δ^{k+1}, ω^{k+1}

Algorithm 4 POSTPROCESS: LS–backtracking

Inputs: The current approximate solution u^k, nonlinear operator $\mathcal{G}$, residual vector f^k, inexact Newton step δ^k, current forcing term ω^k, termination tolerance η.

Outputs: A new solution u^{k+1} and a new forcing term ω^{k+1} or a `flag` if fails to converge.

1. $t = 10^{-4},\ \tau = 1,\ \omega = \omega^k,\ \ell = 10,\ [\gamma_{\min}, \gamma_{\max}] = [0.1, 0.5],\ \omega_{\max} = 0.1$ ▷ initialization
2. $\omega_s = 0.9\left(\omega^k\right)^2$ ▷ safeguard
3. **for** $i = 1, 2, \ldots, \ell$ ▷ finite times of backtracking
4. $\tilde{u} = u^k + \tau\delta^k$ ▷ a trial step
5. **if** $\|\mathcal{G}(\tilde{u})\| \leq (1 - t(1-\omega))\|f^k\|$ **then** ▷ sufficient decrease
6. $\tilde{\delta}^k = \tau\delta^k,\ u^{k+1} = u^k + \tilde{\delta}^k,\ f^{k+1} = \mathcal{G}(u^{k+1})$ ▷ update
7. $\omega^{k+1} = 0.9\dfrac{\|f^{k+1}\|^2}{\|f^k\|^2}$ ▷ initial forcing term $\omega^0 = 0.01$
8. **if** $\omega_s > 0.1$ **then**
9. $\omega^{k+1} = \max(\omega^{k+1},\ \omega_s)$ ▷ safeguard for too small forcing term
10. **end if**
11. $\omega^{k+1} = \max\left(\omega^{k+1},\ \dfrac{\eta}{2\|f^{k+1}\|}\right)$ ▷ safeguard for oversolving
12. $\omega^{k+1} = \min\left(\omega^{k+1},\ \omega_{\max}\right)$ ▷ safeguard for too much inexactness
13. **return** u^{k+1}, ω^{k+1}
14. **end if**
15. Construct the quadratic polynomial $p(\gamma)$ s.t. $p(0) = g(0),\ p'(0) = g'(0),\ p(1) = g(1)$, where $g(\gamma) = \|\mathcal{G}(u^k + \gamma\tau\delta^k)\|$.
16. Find $\hat{\gamma}$ which minimizes $p(\gamma)$ over interval $[\gamma_{\min}, \gamma_{\max}]$.
17. $\tau = \hat{\gamma}\tau,\ \omega = 1 - \hat{\gamma}(1-\omega)$ ▷ backtracking
18. **end for**
19. **return** `flag`: failure of backtracking.

Algorithm 5 POSTPROCESS: TR–contravariant

Inputs: The current approximate solution u^k, nonlinear operator $\mathcal{G}$, residual vector f^k, inexact Newton step δ^k, current forcing term ω^k, residual r^k of GMRES, contraction factor Θ^{k-1} and contravariant Kantorovich quantity h^{k-1} from the previous outer iteration.

Outputs: A new solution u^{k+1} and a new forcing term ω^{k+1} or a `flag` if fails to converge.

$\mu_{\min} = 10^{-6},\ \omega_{\max} = 0.1,\ \omega_{\min} = 10^{-5},\ \rho = 0.9$ ▷ initialization
if $k \geq 1$ **then**
  $\mu = \min\left(1,\ \dfrac{1}{(1+\omega^k)\Theta^{k-1}h^{k-1}}\right)$ ▷ prediction for μ
else
  $\mu = 0.1$
end if
`acceptStep` $=$ `FALSE`, `reduced` $=$ `FALSE`
while ¬`acceptStep` **do**
  if $\mu < \mu_{\min}$ **then**
    return `flag`: regularity test fails.
  else
    $\hat{u}^k = u^k + \mu\delta^k,\ \hat{f}^k = \mathcal{G}(\hat{u}^k)$ ▷ a trial step
    $\Theta^k = \dfrac{\|\hat{f}^k\|}{\|f^k\|}$ ▷ contraction factor
    $h^k = \dfrac{2\|\hat{f}^k - (1-\mu)f^k - \mu r^k\|}{\mu^2\left(1-\left(\omega^k\right)^2\right)\|f^k\|}$ ▷ Kantorovich quantity
    if $\Theta^k \geq 1 - \frac{\mu}{4}$ **then** ▷ no contraction
      $\mu = \min\left(\dfrac{1}{(1+\omega^k)h^k},\ \dfrac{\mu}{2}\right)$, `reduced` $=$ `TRUE` ▷ damping
    else
      $\hat{\mu} = \min\left(1,\ \dfrac{1}{(1+\omega^k)h^k}\right)$
      if $\hat{\mu} \geq 4\mu$ & ¬`reduced` **then**
        $\mu = \hat{\mu}$ ▷ try for a larger step
      else
        `acceptStep` $=$ `TRUE` ▷ step accepted
      end if
    end if
  end if
end while
$\tilde{\delta}^k = \mu\delta^k,\ u^{k+1} = u^k + \tilde{\delta}^k,\ f^{k+1} = \mathcal{G}(u^{k+1})$ ▷ update
$\hat{h} = \dfrac{2\rho\left(\Theta^k\right)^2}{(1+\rho)\left(1-\left(\omega^k\right)^2\right)}$ ▷ a-posterior estimate
$\omega^{k+1} = \min\left(\dfrac{\sqrt{1+\left(\hat{h}\right)^2}-1}{\hat{h}},\ \omega_{\max}\right)$ ▷ quadratic convergence mode, $\omega^0 = 10^{-3}$
$\omega^{k+1} = \max\left(\omega^{k+1}, \omega_{\min}\right)$ ▷ safeguard for oversolving
return u^{k+1}, ω^{k+1}